Fingerprint recognition

Diefenderfer, Graig T.

Monterey California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

FINGERPRINT RECOGNITION

by

Graig T. Diefenderfer

June 2006

Thesis Advisor: Monique P. Fargues
Second Reader: Roberto Cristi

Approved for public release; distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE June 2006	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE Fingerprint Recognition	5. FUNDING NUMBERS
6. AUTHOR(S) Graig T. Diefenderfer	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

The use of biometrics is an evolving component in today's society. Fingerprint recognition continues to be one of the most widely used biometric systems. This thesis explores the various steps present in a fingerprint recognition system. The study develops a working algorithm to extract fingerprint minutiae from an input fingerprint image. This stage incorporates a variety of image pre-processing steps necessary for accurate minutiae extraction and includes two different methods of ridge thinning. Next, it implements a procedure for matching sets of minutiae data. This process goes through all possible alignments of the datasets and returns the matching score for the best possible alignment. Finally, it conducts a series of matching experiments to compare the performance of the two different thinning methods considered. Results show that thinning by the central line method produces better False Non-match Rates and False Match Rates than those obtained through thinning by the block filter method.

14. SUBJECT TERMS Biometrics, Fingerprint, Minutiae, Thinning, Matching	15. NUMBER OF PAGES 153
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

FINGERPRINT RECOGNITION

Graig T. Diefenderfer
Ensign, United States Navy
B.S., United States Naval Academy, 2005

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2006

Author: Graig T. Diefenderfer

Approved by: Monique P. Fargues
Thesis Advisor

Roberto Cristi
Second Reader

Jeffrey B. Knorr
Chairman, Department of Electrical and
Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The use of biometrics is an evolving component in today's society. Fingerprint recognition continues to be one of the most widely used biometric systems. This thesis explores the various steps present in a fingerprint recognition system. The study develops a working algorithm to extract fingerprint minutiae from an input fingerprint image. This stage incorporates a variety of image pre-processing steps necessary for accurate minutiae extraction and includes two different methods of ridge thinning. Next, it implements a procedure for matching sets of minutiae data. This process goes through all possible alignments of the datasets and returns the matching score for the best possible alignment. Finally, it conducts a series of matching experiments to compare the performance of the two different thinning methods considered. Results show that thinning by the central line method produces better False Non-match Rates and False Match Rates than those obtained through thinning by the block filter method.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First, I would like to thank Professor Fargues for devoting the time in leading me through this lengthy process. I would also like to thank Professor Cristi for being the second reader for this thesis. Next, I want to thank the U.S. Naval Academy Electrical Engineering department for sparking my interest in this subject matter. Finally, I want to thank the Misfits and Team Ensign for making my time in Monterey quite enjoyable.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Biometrics is the study of automatically recognizing humans by means of inherently unique physical or behavioral characteristics. Currently, one of the most widespread biometrics is fingerprint recognition. The study of fingerprint attributes dates back to the 1600s, and the first automated fingerprint recognition systems began to be developed in the 1960s. Even so, the application of fingerprint recognition continues to expand in our society. From laptop computers to office buildings, these systems are used as a convenient way of restricting access to authorized users.

A fingerprint is comprised of a pattern of lines, known as ridges. The spaces between individual ridges are referred to as valleys. As a ridge progresses, it can either come to an end, or it can split into two ridges. The location where a ridge comes to an end is known as a termination, and the location where a ridge divides into two separate ridges is called a bifurcation. Terminations and bifurcations are the two basic types of minutiae, which are the points of interest within a fingerprint. By detecting the minutiae in a fingerprint, an effective matching process can be implemented.

The main objective of this research is to create a fingerprint recognition system using the MATLAB environment. Essentially, this study involves reliably extracting minutiae from a fingerprint image and comparing this information to previously defined minutiae data. The

overall process is split into three primary steps: image pre-processing, minutiae extraction, and minutiae matching.

- Image Pre-Processing

The first step involves converting the original grayscale image to a black-and-white image. Known as binarization, this step applies a threshold to the pixels in the grayscale image, where any pixel with a value above a user-specified threshold gets assigned a value of one, whereas any pixel with a value below the threshold gets assigned a value of zero. To ensure minutiae details extracted in later steps are accurate, the binarization process requires careful selection of a threshold value that does not eliminate ridge information and does not produce false ridge structures.

The next phase in the image pre-processing stage involves thinning, i.e., reducing the width of each ridge to one pixel. Two different thinning methods were examined in this research. The first is known as block filter thinning and was developed from scratch in the study. This method preserves pixels along the outer boundaries of the ridges by deleting those pixels that lie within a three-by-three pixel block beneath the outer pixels on each ridge. There are several stages in this thinning phase designed to remove spurious segments generated during the initial block filtering process. Meanwhile, the second thinning method considered in the study attempts to thin the ridges to their central lines only, thereby ensuring that no ridge information has been lost in the thinning process.

- Minutiae Extraction

The second step in the overall fingerprint recognition process involves extracting the minutiae information from the images. The minutiae extraction process is applied to the image with ridges reduced to a width of one pixel and unwanted noise removed. This detection step involves scanning the image to locate fingerprint terminations and bifurcations. The next stage in the minutiae extraction process involves computing the orientation angle for each identified minutia. Specifically, the orientation angle for a termination is defined as the angle between the horizontal and the ridge direction as it ends. The orientation angle for a bifurcation is defined as the angle between the horizontal and the direction of the original ridge moving away from the divergence.

At that point, only the minutiae information needs to be stored for later matching. Original fingerprint images are no longer needed, which significantly decreases storage requirements. In this study, the minutiae information is stored matrix-wise, with each row representing a different minutia. Column one and two of this matrix represent the row and column position of the minutiae within the image, respectively. Column three stores the orientation angle of the minutiae, and column four indicates the type of minutiae.

- Minutiae Matching

The third step in the overall fingerprint recognition process involves matching sets of minutiae data. During the recognition phase, an input image is processed and its minutiae information matched against fingerprint minutiae

information from authorized users, known as templates. Note that there is likely to be some degree of rotational or displacement difference between the input and template image. The processing scheme considered in this study takes this into account and implements the matching processing in a polar coordinate system. By doing so, the robustness of the verification scheme is increased to handle displacements between input and template images.

The next process in this stage involves computing matching scores between input and template images, which represent the similarity between the two sets of data. In this system, the matching score ranges from zero to one, where a matching score of zero and one represent a complete mismatch and perfect match between the input and template, respectively. This matching score is compared to the system threshold to arrive at the final decision. The input and template fingerprints are determined to be from the same finger when the matching score is greater than the threshold. Conversely, the input and template fingerprints are determined to be from different fingers when the matching score is less than the threshold.

- Simulation

A database of fingerprints was generated using the SFINGE software to test the overall system. For each fingerprint, a template was created by sending the image through the previous discussed steps until the matrix of minutiae data was obtained. Also, a different impression for each fingerprint was generated by performing horizontal shifts left and right, vertical shifts up and down, and clockwise and counterclockwise rotations.

Creating the different impressions simulates variable finger alignments in successive login attempts. A series of matching processes using the two thinning methods was conducted between the generated images and the templates. Results show that the central line thinning produces better overall results, and that the block filter thinning method's performance degraded significantly when dealing with rotated input images. For most rotated input images, using the block filter thinning method caused the system to incorrectly conclude that two fingerprints from the same finger were from different fingers. The central line thinning method, however, worked well for all variations of input images. This shows that this method is rotational invariant and will accurately locate the minutiae regardless of the input alignment. Therefore, the central line thinning method should be used for the thinning process in a real world system.

As a whole, the developed fingerprint recognition system works well. From ridge thinning, to minutiae extraction, to minutiae matching, the system produces reliable results. In the coming years, biometrics will continue to play a larger role in society, and fingerprint recognition will likely remain at the center of this expansion.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The use of biometrics continues to evolve in many areas of society. Fingerprint readers can be found on laptop computers, iris scanners are being installed at locations of heightened security, and voice recognition software is being incorporated into automobiles. Whatever the reason for the biometric system, it is evident that their use will continue to develop during the coming years. This chapter presents an overview of the basic concepts behind biometric systems.

A. BACKGROUND

In order for a human physiological or behavioral characteristic to be used as a beneficial biometric trait, it must satisfy four criteria. First of all, it needs to be universal, with each person in possession of the given characteristic. Secondly, it should be a distinctive quality, meaning there should be a significant distinction in the characteristic between any two given persons. Next, there needs to be a certain permanence inherent to the feature, i.e., the measured elements should remain relatively invariable over a period of time. Last of all, the attribute should be easy to collect and measured quantitatively [Jain, Ross, & Prabhakar, 2004]. Other issues, such as performance, acceptability, and circumvention, need to be examined for a system that implements personal recognition with biometrics. The performance of a system concerns the accuracy and speed of recognition. Meanwhile, the acceptability of a system refers to the willingness of the general population to allow use of a particular biometric in everyday life.

Finally, circumvention deals with how easy it is to fool the system through spoofing. Therefore, a good biometric system will have high accuracy and speed, be widely accepted among the public, and have high resistance to fraudulent attacks [Jain, Ross, & Prabhakar, 2004].

There are two main operating modes for biometric systems. The first, and simplest mode, is called verification. Here, it is necessary for the person to claim an identity through an identification number, user name, or other means. The system then gathers the input data and compares it to the template data previously stored for that person. This comparison is a one-to-one comparison, and the system is only trying to verify that the person attempting to gain access is truly who he claims to be. If the input data does not match the template data, the system will deny access. The second operating mode is called identification. In this mode, the system will compare the input data to all sets of template data already stored. This operation is a one-to-many comparison, and there is no need for the person to claim an initial identity. If the input data matches any of the template data sets, the system will allow access [Jain, Ross, & Prabhakar, 2004]. Note that the identification mode is more computationally intensive than verification mode since it requires conducting a comparison with each template.

B. OBJECTIVE

The objective of this research is to develop a fingerprint recognition system using MATLAB. Beginning with an input image, the system processes the data and collects the identifying features of the fingerprint. Next, it compares this information to previously stored

information from various fingerprints. After making the comparison, the system determines if the input image matches the data of a fingerprint already in the database. A few different processing methods are used to extract the identifying features, and the performance of each technique is analyzed.

C. THESIS ORGANIZATION

The remainder of this thesis provides a detailed description of the research conducted. Chapter II presents background information on biometrics in general, while Chapter III focuses specifically on fingerprints. Next, Chapter IV provides a detailed account of the developed minutiae detection process. From here, Chapter V discusses the minutiae matching process used in this system. Chapter VI features the experimental results obtained from the complete system. Finally, conclusions and recommendations are presented in Chapter VII.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BIOMETRICS

This chapter introduces a variety of common biometric systems in use today or currently being developed. It then proceeds to discuss some of the most important issues concerning biometrics in general.

A. SYSTEMS IN USE TODAY

1. Fingerprint

Fingerprint recognition has been present for a few hundred years. Nowadays, the technology in this area has reached a point where the cost of purchasing a fingerprint scanner is very affordable. For this reason, these systems are becoming more widespread in a variety of applications. As seen in Figure 2.1, Cell phones, PDAs, and personal computers are a few examples of items incorporating fingerprint recognition to increase user security.

Figure 2.1. Fingerprint sensors in everyday products. (From: [Mainguet, 2006])

Fingerprint systems are generally best utilized in verification systems or small-scale identification systems because a large-scale identification system requires extensive computational resources under current products. In addition, a large system would undoubtedly encounter some fingerprints that are unsuitable for use, due to cuts

or other defects [Jain, Ross, & Prabhakar, 2004]. Therefore, cell phones and computers, which both potentially have a small number of users, are ideal products for this technology.

2. Iris

Iris recognition has taken on greater interest in recent years. As this technology advances, purchasing these systems has become more affordable. These systems are attractive because the pattern variability of the iris among different persons is extremely large. Thus, these systems can be used on a larger scale with a small possibility of incorrectly matching an imposter. Also, the iris is well protected from the environment and remains stable over time. In terms of localizing the iris from a face, its distinct shape allows for precise and reliable isolation [Daugman, 2004]. Figure 2.2 shows the unique iris pattern data extracted from a sample input.

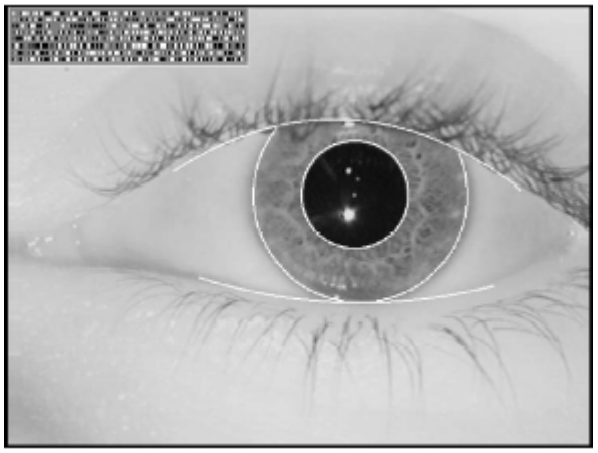


Figure 2.2. Example of an iris pattern. (From: [Daugman, 2004])

The accuracy and speed of current iris systems allows the possibility of implementing this technique in a large-scale system. The iris of each person is distinctive, and

identical twins even have different patterns. Since it is extremely difficult to alter the texture of the iris through surgery, it would be difficult for someone to circumvent the system. Along the same line, it is relatively easy for the system to detect when an artificial iris, such as a specially made contact lens, is being used in an attempt to gain access [Jain, Ross, & Prabhakar, 2004]. Thus, as time goes on, it is likely that iris recognition systems will be widely used in many areas of society.

3. Voice

Voice recognition offers a dynamic range of processing possibilities. Unlike fingerprint and iris recognition, which are limited to a few techniques, voice recognition has a variety of methods for implementation. Specifically, this flexibility is evident in a system that forces a user to speak a phrase that is different for each attempt. This versatility makes it much more difficult for someone to spoof the system. At the same time, it requires the system to have a more advanced detection algorithm [Faundez-Zanuy & Monte-Moreno, 2005].

Another reason voice-based systems are dynamic is because voice is a combination of physiological and behavioral biometrics. The physiological component is governed by physical characteristics, which are invariant for an individual, while the behavioral element can change over time. This combination offers a wider range of possibilities among the general population. A phone-based speaker recognition system provides a practical application, as it allows for recognition from a remote location, where collection of other biometric data may not

be possible. Speech features, however, are sensitive to background noise, different microphones, and degradation over a communication channel, and care should be taken to ensure the quality of the signal remains at an acceptable level [Jain, Ross, & Prabhakar, 2004].

4. Face

There have been significant achievements in the face recognition field over the past few years. Thanks to these advancements, this problem appears to eventually become technologically feasible as well as economically realistic. Several companies now offer face recognition software that can produce high-accuracy results with a database of over 1000 people. In addition, current research involves developing more robust approaches that accounts for changes in lighting, expression, and aging, where potential variations for a given person are illustrated in Figure 2.3. Also, other problem areas being investigated include dealing with glasses, facial hair, and makeup [Pentland & Choudhury, 2000].

Figure 2.3. Facial image variations amongst the same subject. (From: [Gao & Leung, 2002])

A facial recognition system has numerous advantages over other biometric systems. First of all, the system can be unobtrusive, operating at a large distance from the subject. Also, it does not require the person to have a set interaction with the system. The camera only needs to capture a useable image of the face. Next, the system is

usually passive and can operate on fairly low power. Finally, a face recognition system would probably be widely accepted by the general public. Since we perform facial recognition in our daily life, an automated system that performs the same task is likely to have more support than other, more intrusive biometrics [Pentland & Choudhury, 2000].

5. Gait

Gait-based recognition involves identifying a person's walking style. Although these systems are currently very limited, there is a significant amount of research being conducted in this area. At this time, however, it is unknown if the obtainable gait parameters provide enough discrimination for a system to be applied on a large scale. Furthermore, studies have shown that gait changes over time and is also affected by clothes, footwear, walking surfaces, and other conditions. Figure 2.4 outlines the various stages of a gait cycle. [Boulgouris, Hatzinakos, & Plataniotis, 2005].

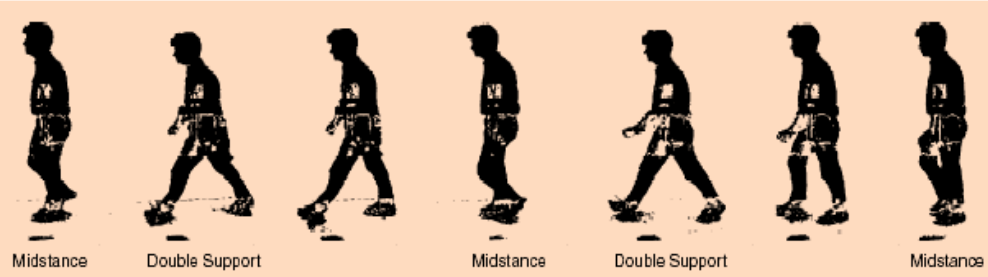


Figure 2.4. Samples recorded from a gait cycle. (From: [Boulgouris, Hatzinakos, & Plataniotis, 2005])

Acquisition of gait is similar to the process used for acquiring facial images and can be performed from a distance without requiring the subject's cooperation. As a result, this system can be implemented covertly for

enhanced security. Furthermore, it is difficult to hide or fake one's gait, particularly when the presence of the system is unknown. A disadvantage to gait recognition is that it uses a sequence of video footage, making the process input intensive and computationally expensive [Jain, Ross, & Prabhakar, 2004].

6. Hand Geometry

Hand geometry systems are one of the most basic biometric systems in use today. A two-dimensional system can be implemented with a simple document scanner or digital camera, as these systems only measure the distances between various points on the hand. Meanwhile, a three-dimensional system provides more information and greater reliability. These systems, however, require a more expensive collection device than the inexpensive scanners that can be used in a two-dimensional system. An example of a commercial three-dimensional scanner is shown in Figure 2.5. As seen in this image, the physical size of the scanner limits its application in portable devices.

Figure 2.5. Commercial three-dimensional scanner. (From: [Faundez-Zanuy, 2005])

The primary advantage of hand geometry systems is that they are simple and inexpensive to use. Also, poor weather and individual anomalies such as dry skin or cuts along the

hand do not appear to negatively affect the system. The geometry of the hand, however, is not a very distinctive quality. In addition, wearing jewelry or other items on the fingers may adversely affect the system's performance. Finally, personal limitations in dexterity due to arthritis or other medical condition also have a negative impact. Therefore, these systems generally should not be used in a large-scale environment [Jain, Ross, & Prabhakar, 2004].

7. Multimodal

Multimodal systems employ more than one biometric recognition technique to arrive at a final decision. These systems may be necessary to ensure accurate performance in large dataset applications. Combining several biometrics in one system allows for improved performance as each individual biometric has its own strengths and weaknesses. Using more than one biometric also provides more diversity in cases where it is not possible to obtain a particular characteristic for a person at a given time. Although acquiring more measurements increases the cost and computational requirements, the extra data allows for much greater performance [Ko, 2005].

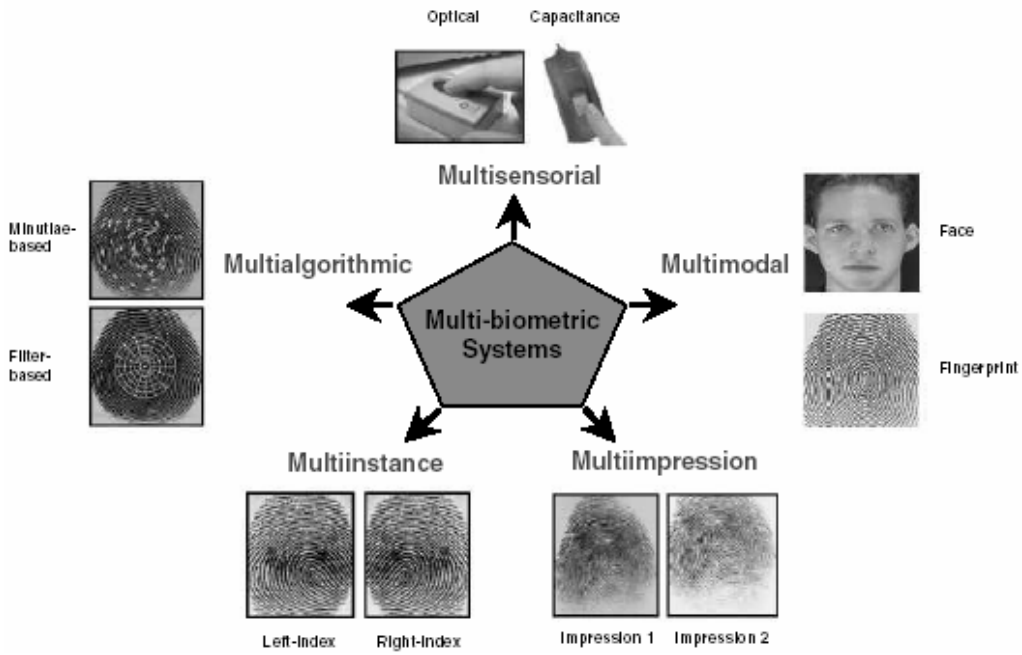


Figure 2.6. Multibiometric categories. (From: [Ko, 2005])

Similar to multimodal systems, there are several other techniques aimed at improving the performance of a biometric system, as outlined in Figure 2.6. Multi-algorithmic techniques acquire a single sample from one sensor and process this signal with two or more different algorithms. Multi-instance systems use a sensor to obtain data for different instances of the same biometric, such as capturing fingerprints from different fingers of the same person. Multi-sensorial systems sample the same biometric trait with two or more different sensors, such as scanning a fingerprint using both optical and capacitance scanners [Ko, 2005].

B. ISSUES WITH BIOMETRICS

1. Security

Maintaining the integrity of a biometric system is a critical issue. This concept deals with ensuring that the input to the system is in fact presented by its legitimate

owner, and also assuring that the input pattern is matched with an authentic template pattern. There are numerous ways someone can attempt to bypass a biometric system, all of which endeavor to attack a weakness in the system. The primary weaknesses in a system arise from the fact that biometrics are not secrets. In addition, the biometric characteristics of a person can not be altered. When a biometric identifier has been compromised, an attacker can use this information to develop a fraudulent dataset to fool the system into accepting his access. As a result, the algorithm may not be able to detect falsified data when the matching process is completely autonomous and without human monitoring [Jain, Pankanti, Prabhakar, Hong, & Ross, 2004].

Along the same lines, once someone's biometric information has been compromised, it is not possible to change this information. Unlike a password or PIN number, it is not possible for someone to alter his natural characteristics. For example, fingerprints remain the same throughout an individual's lifetime and can not be changed when they have been accessed and used for fraudulent purposes. A current technique to combat fraudulent claims is known as liveness detection. In this technique, the system can determine if the input measurements are originating from an inanimate object instead of the actual person [Jain, Pankanti, Prabhakar, Hong, & Ross, 2004].

2. Privacy

Another important issue in the biometric field concerns maintaining privacy. Whenever a person has a biometric identifier recorded, the individual loses some anonymity that cannot be recovered. Critics of biometrics

envision this information being used in coming years by the government to monitor actions and behaviors of all citizens. While this idea is not likely to be implemented in the near future, its possibility leads many to be cautious of giving up biometric identifiers. Also, it is possible that some biometrics capture more information than one's identity. Some schemes may provide additional information, such as a person's health and medical history. Although this idea is currently undeveloped, its potential impact would likely invoke concern among the public [Woodward, 1997].

Nevertheless, biometrics can be used as a highly effective way to maintain individual privacy despite the negative issues associated with their uses. If one's personal information can only be accessed through a biometric matching process, this information will remain much safer than if its access were controlled by a standard password. Similarly, the possibility of a thief being able to use a stolen card would greatly diminish if a credit card could only be used when supplied with the correct fingerprint. Therefore, these advantages tend to outweigh the concerns over losing a degree of personal privacy by providing biometric data. In addition, many products today do not even store the template data in its original form. Instead, the data is stored in an encrypted form that is only recoverable by that specific system to prevent an attacker from using the template information in its own system [Jain, Ross, & Prabhakar, 2004].

3. Accuracy

There are two basic types of errors that occur in a biometric system. The first error occurs when the system

mistakes measurements from two different persons to be from the same person. This is called a false match, or sometimes termed a false acceptance. The second type of error is known as a false non-match and happens when the system mistakes measurements from the same person to be from two different persons. This scenario is also referred to as a false rejection [Jain, Ross, & Prabhakar, 2004].

In terms of probabilistic quantities, the False Match Rate (FMR) is the probability that the system will decide to allow access to an imposter. Meanwhile, the False Non-match Rate (FNMR) represents the probability that the system denies access to an approved user. In each biometric system, there is a tradeoff between the FMR and FNMR because both of these values are functions of what is called the “system threshold”. When a system makes a comparison between an input image and an image already stored in the database, it calculates a matching score for the two images. This matching score provides a numerical value of the similarity between the two images. A high matching score indicates high similarity, whereas a low matching score indicates low similarity between the images. The matching score is compared to the system threshold, which is a predefined level in the system. The images are classified as a match when the matching score is greater than the system threshold, and a non-match is produced when the matching score is less than the system threshold. Modifying the threshold has an effect on the performance of the system. When the threshold is decreased, meaning there is a less stringent matching score, the FMR will increase while the FNMR decreases. In such cases, more imposter matches will occur since it is easier to produce a match

with the lower threshold. At the same time, the lower threshold allows for fewer false non-matches for approved users. Similarly, as the threshold increases, the FMR decreases while the FNMR increases. This would be the case for a high security application, where a system is setup to greatly reduce the number of false matches. As a side effect, the increased threshold causes more false non-matches due to the stricter matching requirements.

The performance of a system is commonly expressed in a receiver operating characteristic (ROC) curve. In this graph, the FMR is plotted against the FNMR of the system for various threshold values. As shown in Figure 2.7, as the FNMR increases, the FMR of the system decreases. Conversely, decreasing the FNMR results in an increase to the FMR [Jain, Ross, & Prabhakar, 2004].

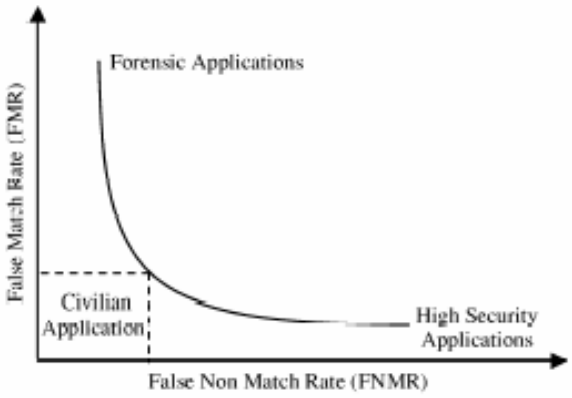


Figure 2.7. Receiver Operating Characteristic (ROC) curve. (From: [Jain, Ross, & Prabhakar, 2004])

4. Scale

A final issue concerning biometric systems is the number of individuals enrolled in the database. For each individual, there is a unique set of template data that corresponds to that person's identity. In a verification system, the number of people enrolled does not have a

significant impact because the match is performed solely on a one-to-one basis. In an identification system, however, the matching process is more complex and is greatly affected by the number of enrollees as the system must perform a one-to-one comparison for each set of template data. For a very large system, this process can take a significant amount of time. One solution to reduce this time involves categorizing the template data based on a significant pattern. In the case of a fingerprint system, the initial categorization may divide the templates into classes based on their global patterns. Although this procedure is difficult to implement in practice, it serves as a foundation for reducing the scale of the system (Jain, Pankanti, Prabhakar, Hong, & Ross, 2004).

C. CONCLUSION

Biometric applications continue to evolve in all areas of society, and care needs to be taken to ensure a specific biometric is suited for a given application. More complex biometrics, such as iris and fingerprint recognition, may be better suited for a large-scale application than a more basic biometric would, such as hand geometry. Meanwhile, it is still unknown what capabilities will be obtainable through biometrics currently under development, specifically face and gait recognition. In any case, all biometrics raise issues which need to be examined. The following chapter will introduce the foundations of one the most widespread biometrics: the fingerprint.

THIS PAGE INTENTIONALLY LEFT BLANK

III. FINGERPRINT MATCHING

This chapter presents a brief history of the evolution of fingerprint identification. It also provides the basic information regarding the composition of a fingerprint. Finally, it discusses a variety of matching techniques used today.

A. HISTORY

Fingerprints have been scientifically studied for a number of years in our society. The characteristics of fingerprints were studied as early as the 1600s. Meanwhile, using fingerprints as a means of identification first transpired in the mid-1800s. Sir William Herschel, in 1859, discovered that fingerprints do not change over time and that each pattern is unique to an individual. With these findings, he was the first to implement a system using fingerprints and handprints to identify an individual in 1877. At the time, his system was a simple one-to-one verification process. By 1896, police forces in India realized the benefit of using fingerprints to identify criminals, and they began collecting the fingerprints of prisoners along with their other measurements [International Biometric Group, 2003].

With a growing database of fingerprint images, it soon became desirable to have an efficient manner of classifying the various images. Between 1896 and 1897, Sir Edward Henry developed the Henry Classification System, which quickly found worldwide acceptance within a few years. This system allows for logical categorization of a complete set of the ten fingerprint images for a person. By establishing groupings based on fingerprint pattern types,

the Henry System greatly reduces the effort of searching a large database. Until the mid-1990s, many organizations continued to use the Henry Classification System to store their physical files of fingerprint images [International Biometric Group, 2003].

As fingerprints began to be utilized in more fields, the number of requests for fingerprint matching began to increase on a daily basis. At the same time, the size of the databases continued to expand with each passing day. Therefore, it soon became difficult for teams of fingerprint experts to provide accurate results in a timely manner. In the early 1960s, the FBI, Home Office in the United Kingdom, and Paris Police Department began to devote a large amount of resources in developing automatic fingerprint identification systems. These systems allowed for an improvement in operational productivity among law enforcement agencies. At the same time, the automated systems reduced funding requirements to hire and train human fingerprint experts. Today, automatic fingerprint recognition technology can be found in a wide range of civilian applications [Maltoni, Maio, Jain, & Prabhakar, 2003].

B. FINGERPRINT DETAILS

A fingerprint pattern is comprised of a sequence of ridges and valleys. In a fingerprint image, the ridges appear as dark lines while the valleys are the light areas between the ridges. A cut or burn to a finger does not affect the underlying ridge structure, and the original pattern will be reproduced when new skin grows. Ridges and valleys generally run parallel to each other, and their patterns can be analyzed on a global and local level. At

the global level, the fingerprint image will have one or more regions where the ridge lines have a distinctive shape. These shapes are usually characterized by areas of high curvature or frequent ridge endings and are known as singular regions. The three basic types of these singular regions are loop, delta, and whorl, examples of which are shown in Figure 3.1. Many matching algorithms use the center of the highest loop type singularity, known as the core, to pre-align fingerprint images for better results. As shown in Figure 3.2, these three basic singularities help form the five major classes of fingerprints [Maltoni, Maio, Jain, & Prabhakar, 2003].

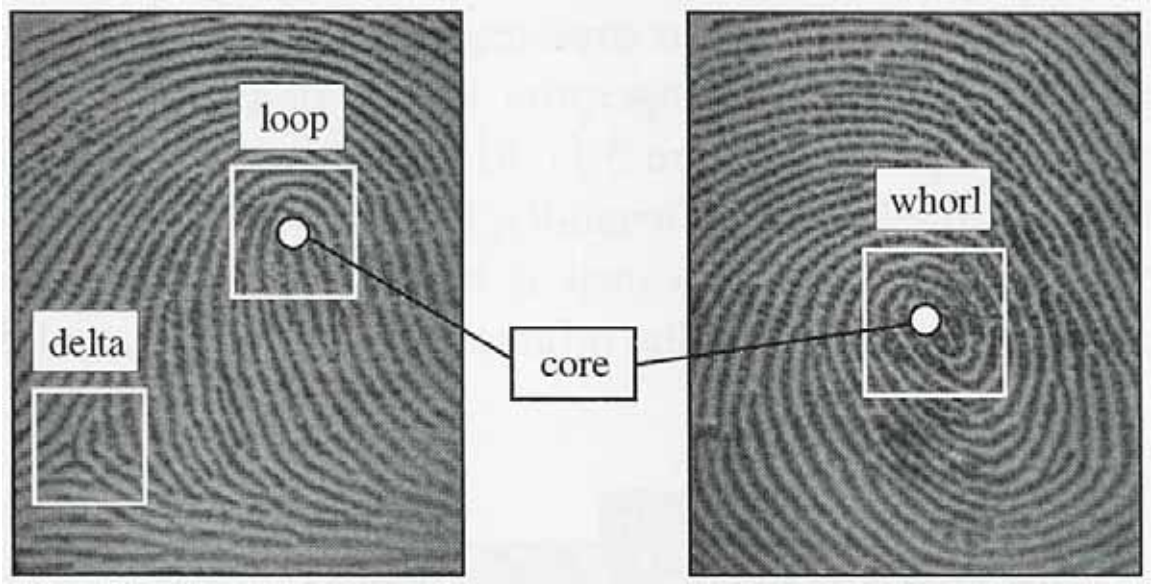


Figure 3.1. Singular regions and core points. (From: [Maltoni, Maio, Jain, & Prabhakar, 2003])

Figure 3.2. Examples of fingerprint classes. (From: [Maltoni, Maio, Jain, & Prabhakar, 2003])

While the global level allows for a general classification of fingerprints, analyzing the image at the local level provides a significant amount of detail. These details are obtained by observing the locations where a ridge becomes discontinuous, known as minutiae points. The most common types of minutiae are shown in Figure 3.3. In general, a ridge can either come to an end, which is called a termination, or it can split into two ridges, which is called a bifurcation. The other types of minutiae are slightly more complicated combinations of terminations and bifurcations. For example, a lake is simply a sequence of two bifurcations in opposing directions, while an independent ridge features two separate terminations within a close distance. The FBI minutiae-coordinate model considers only terminations and bifurcations within a fingerprint image. In all, analyzing a fingerprint on the local level provides the necessary information to accurately distinguish one fingerprint from another.

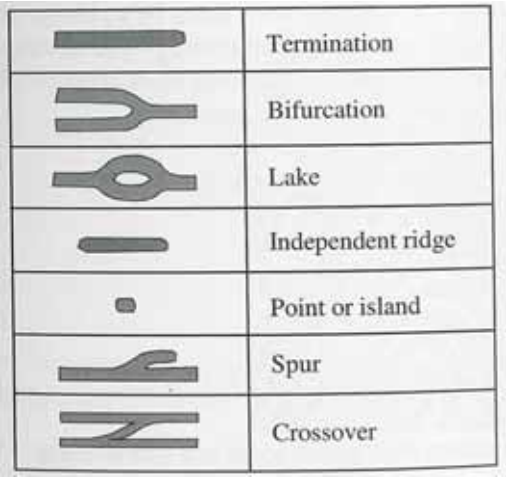


Figure 3.3. Basic types of minutiae. (From: [Maltoni, Maio, Jain, & Prabhakar, 2003])

C. FINGERPRINT MATCHING TECHNIQUES

1. Minutiae-Based

There are several categories of fingerprint matching techniques. One such category employs methods to extract minutiae from the fingerprint images, and then compares this data to the previously stored template data sets. In most cases, the minutiae details are stored as sets of points in the two-dimensional plane. For each minutia, the x- and y-coordinates indicating its location within the image are recorded. Other stored parameters may include the orientation angle of each minutiae as well as the specific type of minutiae located. Generally, minutiae-based methods require a significant amount of pre-processing to produce accurate results [Maltoni, Maio, Jain, & Prabhakar, 2003].

There are a variety of methods in use today for extracting the minutiae from a fingerprint. One method involves thinning the fingerprint image, then performing a scan with a three pixel-by-three pixel block across the entire image. This process will be explained in full detail in the upcoming Chapters. Another method

incorporates a bank of filters in order to extract the minutiae. Specifically, the region of interest gets filtered in eight different directions, which completely captures the local ridge characteristics using a bank of Gabor filters. When the Gabor filters are properly tuned, they are able to remove noise while maintaining the true ridge and valley structures. Since a minutiae point can be considered an anomaly among locally parallel ridges, these points can be detected after applying the bank of Gabor filters [Jain, Prabhakar, Hong, & Pankanti, 2000].

2. Image-Based

Image-based techniques are another significant category of fingerprint matching. These processes are appealing because they do not require a significant amount of pre-processing to produce acceptable results. In most cases, the only pre-processing methods that are applied are a binarization and thinning phase. Therefore, imaged-based techniques have a better computational efficiency than the standard minutiae-based techniques. Also, for low quality fingerprint images, image-based techniques produce better results than minutiae extraction methods, where it may be difficult to reliably extract the actual minutiae points [Seow, Yeoh, Lai, & Abu, 2002].

An important component for image-based matching is dealing with rotation. Since the input image might be oriented differently than the template image, it is necessary to apply a rotational correction to achieve the best results. Many systems superimpose the input image with the template image and compute the correlation between corresponding pixel values for a variety of displacement and rotational values. The maximum correlation value

produced in this process relates to the best possible alignment between the input and the template [Maltoni, Maio, Jain, & Prabhakar, 2003]. A similar technique involves using the phase components from two-dimensional Discrete Fourier Transforms of the images to determine the similarity between the two. If the matching score exceeds the threshold for the system, the input and template are treated as a match [Ito et al., 2005]. In most cases, image-based techniques offer a good alternative when an input image is of poor quality.

Another technique for an image-based fingerprint matching system involves wavelets, where fingerprint patterns are matched based on wavelet domain features. A primary advantage to this approach is that these features can be directly extracted from the fingerprint image without applying any pre-processing steps. Once the core point has been determined, a rectangular region surrounding the core is established, which is referred to as the central sub-image. This area is then divided into non-overlapping square blocks of uniform size. From here, the wavelet decomposition is computed on each block, and its wavelet features are calculated. Next, a global feature vector is formed, which includes the features extracted from each block of the central sub-image. Once the feature extraction has been performed, it is possible to conduct a matching sequence with the template features. The lower computational requirements of this process make using wavelet features attractive to a small-scale system [Tico, Immonen, Rämö, Kuosmanen, & Saarinen, 2001].

3. Ridge Feature-Based

In many images, minutiae extraction may be difficult to conduct in an efficient manner. A low quality image containing a large amount of noise presents problems for minutiae-extracting algorithms. In such a case, other options to acquire meaningful data from a fingerprint become necessary. Analyzing various ridge features provides this versatility. There are several features that are commonly examined in today's systems, ranging from fairly basic to more advanced. At the basic end, the physical size and shape of the external fingerprint silhouette can be computed. Additionally, recording the number, type, and position of singular regions provides further information. Although there is much variation to these numbers, this approach offers some data when little else can be extracted [Maltoni, Maio, Jain, & Prabhakar, 2003].

On a slightly more advanced level, the spatial relationship and geometrical attributes of the ridge lines can be examined. Also, gathering global and local texture information is another option. A final ridge feature that can be analyzed is the location of sweat pores within the ridges. Even though sweat pores are highly discriminant among the population, detecting them requires an advanced collection system, and their presence would most likely be unnoticeable in low quality images. The basic ridge features, however, are obtainable from any quality image. Since minutiae-based methods require an image of good quality, ridge features offer an alternative for poor images. Furthermore, ridge feature-based techniques do not have to be limited to images of poor quality. Instead,

they can be used in conjunction with minutiae-based techniques for images of good quality. With more data to be used in the matching process, the accuracy and robustness of a system would undoubtedly increase [Maltoni, Maio, Jain, & Prabhakar, 2003].

D. CONCLUSION

The use of fingerprints for identification purposes has been present in our society for a number of years. Their characteristics can be analyzed on a global level as well as a local level. While the global characteristics can provide a general classification, it is necessary to analyze a fingerprint on the local level to obtain distinctive classification details. One technique used on the local level involves analyzing the minutiae points within a fingerprint. This process will be examined in further detail in the following chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. MINUTIAE DETECTION

Accurate minutiae detection is an essential component for all minutiae-based fingerprint recognition systems. Without accurate minutiae detection, the results and performance of a system are not reliable. This chapter explores the numerous techniques applied to achieve a dependable minutiae detection system.

A. IMAGE PRE-PROCESSING

It is first necessary to apply several pre-processing steps to the original fingerprint image to produce consistent results in the classic minutiae extraction procedure. Such steps generally include image binarization, noise removal, and thinning. In this thesis, we use the SFINGE software included in Maltoni, Maio, Jain, and Prabhakar (2003) to develop noise-free fingerprint images. This eliminates the need for a sensor noise removal step and enables us to focus on the other steps involved in the minutiae detection process. Note that studies have shown that fingerprint matching algorithms yields very similar performance results when applied to the synthetic fingerprints created by SFINGE as when applied to real fingerprint images [Maltoni, Maio, Jain, & Prabhakar, 2003].

1. Binarization

Image binarization is the process of turning a gray-scale image to a black and white image. In a gray-scale image, a pixel can take on 256 different intensity values while each pixel is assigned to be either black or white in a black and white image. This conversion from gray-scale to black and white is performed by applying a threshold

value to the image. In MATLAB, a value of one means the pixel is white, whereas a value of zero indicates the pixel is black. For a gray-scale image, the pixels are decimal values between zero and one. When a threshold is applied to an image, all pixel values are compared to the input threshold. Any pixel values below the threshold are set to zero, and any values greater than the threshold are set to one. By the end of this process, all pixel values within the image are either zero or one, and the image has been converted to binary format.

A critical component in the binarization process is choosing a correct value for the threshold. If the threshold is set too low, then the resulting binary image will primarily be comprised of white pixels. Conversely, if the threshold is set too high, the resulting image will feature a large number of undesired black pixels. Thus, the threshold must be selected carefully to ensure the data information is preserved after the binarization. The threshold values used in this study were selected empirically by trial and error. In addition, the synthetic fingerprints were generated with the absence of background noise, allowing for a more effective binarization process. An example of this process is shown in Figure 4.1.

Figure 4.1. Results of image binarization.

As seen above, the binarization converts a gray-scale image to a purely black and white image. It should be noted, however, that this process is not perfect, as some of the ridges near the boundary of the image have been turned to white. Although these ridges can be preserved by simply increasing the value of the threshold, this action may also produce highly undesired results. For example, increasing the threshold could change a pixel separating a ridge termination from a neighboring ridge from white to black. This change would then make the termination appear to connect with the neighboring ridge, thus creating a false bifurcation where there should be a termination. Therefore, simply increasing the threshold is not a viable solution to preserve these ridges. Furthermore, the

majority of these imperfections can be removed in a later step since these discontinuities only occur near the perimeter.

2. Thinning

After binarization, another major pre-processing technique applied to the image is thinning, which reduces the thickness of all ridge lines to a single pixel. Following thinning, the location and orientation of the minutiae should still be the same as in the original image to ensure accurate estimation of their locations. There are a variety of thinning methods employed in today's systems. The first technique discussed, which involves thinning along the outer boundary of the ridges via a block filter, was developed from scratch during this study. The second technique is an advanced method originally proposed by Ahmed and Ward (2002) that focuses on thinning the ridges to their central lines.

a. Block Filtering

This thinning method attempts to preserve the outermost pixels along each ridge. First, a border of white pixels is place at the boundaries of the black and white image to ensure accurate implementation of the following steps. As a result, all pixels within the first five rows, last five rows, first five columns, and last five columns are assigned a value of one. The steps in the block filtering process commence following this. Table 4.1 outlines these steps and provides a brief description of the goal for each step.

Step 1: ridge width reduction – reduces width of thick ridges to enable more effective block filtering
Step 2: passage of block filter – right to left and left to right filtering attempts to preserve outer boundaries of ridges
Step 3: removal of isolated noise – removes unwanted segments produced by filtering
Step 4: scan combination – images from right to left and left to right filtering are combined into one image
Step 5: elimination of one pixel from two-by-two squares of black – further thins image following scan combination
Step 6: removal of unwanted spurs – removes short line segments protruding from ridges after scan combination
Step 7: removal of duplicate horizontal and duplicate vertical lines – removes these imperfections produced when scans are combined

Table 4.1. Steps in block filter process.

- Step One: ridge width reduction

This step involves applying a morphological process to the image to reduce the width of the ridges. The two basic morphological processes are erosion and dilation. Dilation is an operation that thickens objects in a binary image, while erosion thins objects in a binary image. In this step, a dilation process is used to thicken the area of the valleys in the fingerprint. As a result of

dilating the valleys, the ridges are effectively eroded. A conservative structuring element consisting of four ones arranged in a two-by-two square is used for the valley dilation to achieve some ridge width reduction while minimizing the amount of discontinuities formed. Thus, the valley dilation causes the width of the ridges to be reduced by a slight amount. Figure 4.2 displays the effect of implementing the valley dilation.

Figure 4.2. Impact of performing valley dilation.

- Step Two: passage of block filter

The next step involves performing a pixel-by-pixel scan for black pixels across the entire image. Note that in MATLAB, image rows are numbered in increasing order

beginning with the very top of the image as row one. Similarly, columns are numbered in increasing order beginning with the leftmost side of the image as column one. This initial scan begins with the pixel in row one and column one, and then continues to move to the adjacent column on the right. When the scanning process reaches the last column of the image, it moves one row down and also resets to the first column and continues until it locates a black pixel. At that point, the scheme turns the pixels contained in a three-by-three box adjacent to the located black pixel to white, beginning with the pixel diagonally down and to the right of the located black pixel. The process is illustrated in Figure 4.3 below.

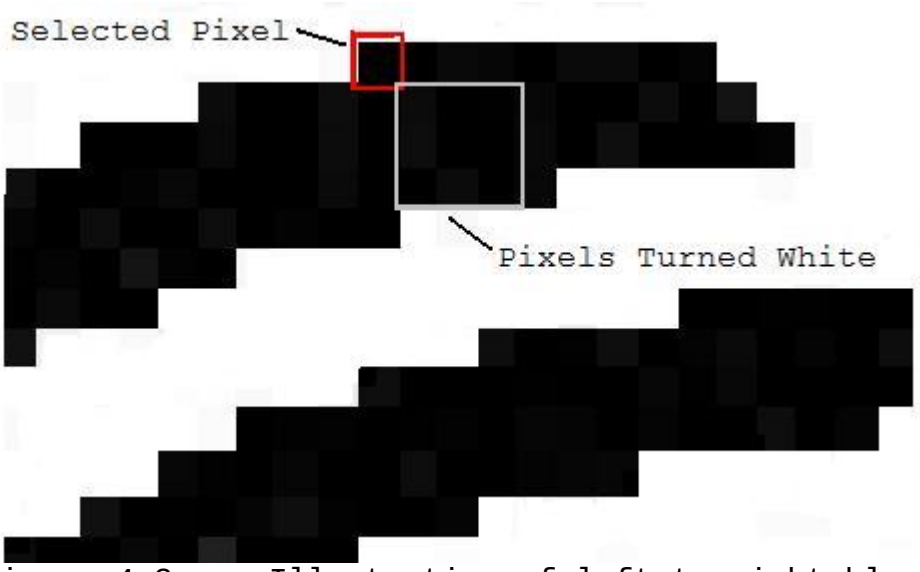


Figure 4.3. Illustration of left to right block filtering on a magnified illustration of a ridge.

The left to right scan continues until it covers the entire image. Next, a similar scan is performed across the image from right to left beginning at the pixel in row one and the last column. This scan moves to the adjacent

column on the left. When the scanning process reaches the first column of the image, it moves one row down and also resets to the last column and continues until it locates a black pixel. Upon detecting a black pixel, the scan stops and turns the pixels contained in a three-by-three box adjacent to the located black pixel to white, beginning with the pixel diagonally down and to the left. Figure 4.4 depicts this process.

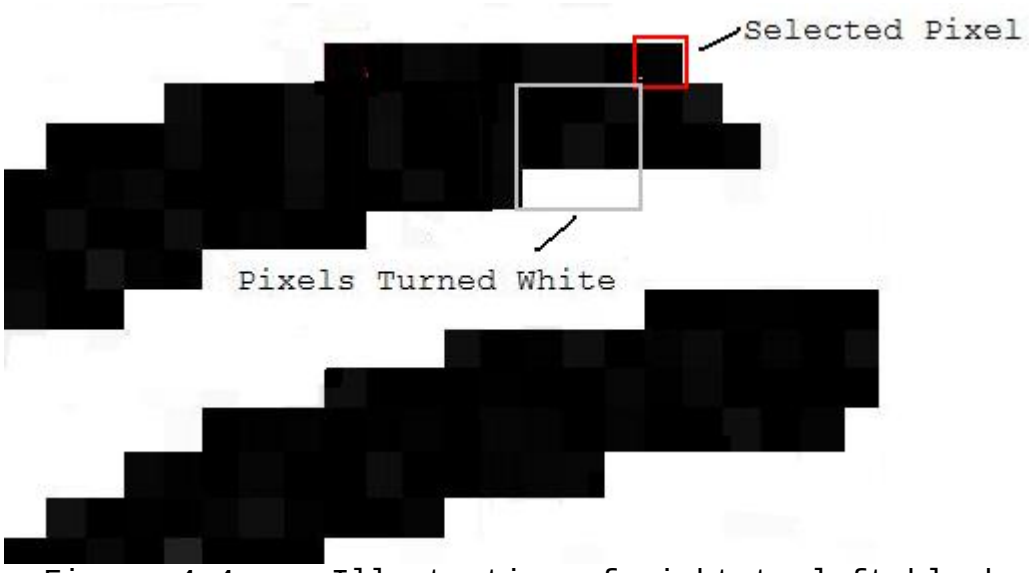


Figure 4.4. Illustration of right to left block filtering on a magnified illustration of a ridge.

The right to left scan continues until all pixels in the image are covered. Note that these two separate scans are needed because one scan by itself does not perform adequately on all regions of the fingerprint. The different types of curvature within a fingerprint have an adverse effect during some portion of the scan. Figure 4.5 shows the results of the two separate scans across a sample fingerprint image.

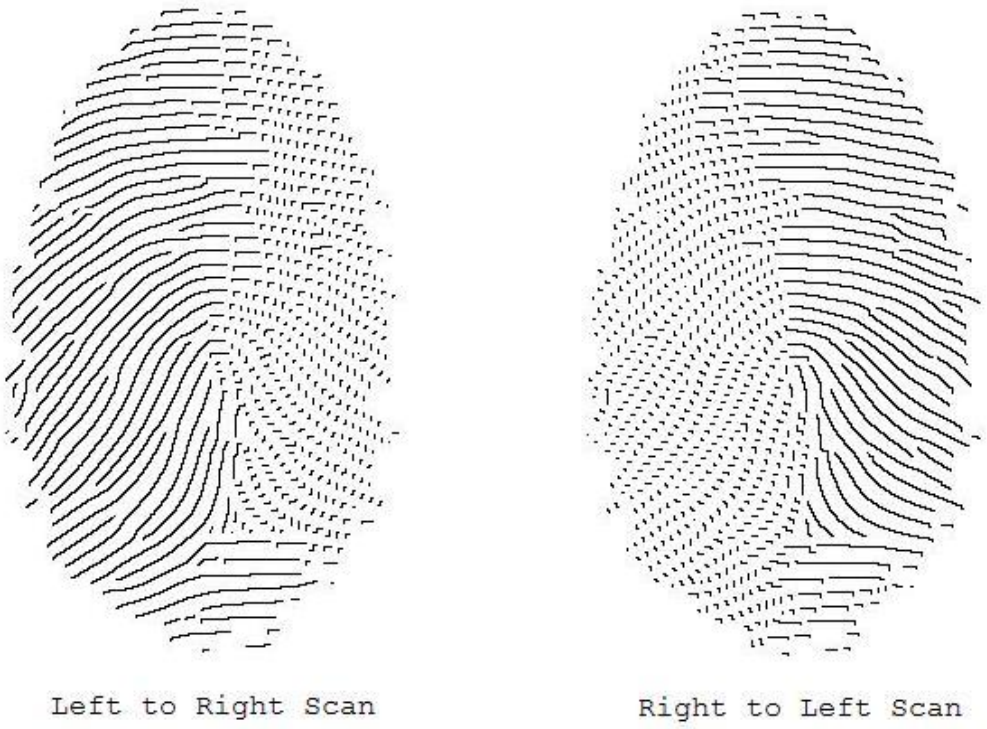


Figure 4.5. Output of filtering in both directions.

Figure 4.5 shows that the left to right scan works well on ridges that move up and to the right, whereas the right to left scan works well on ridges that move down and to the right. In each case, the ridges in the good areas have been reduced to a width of one pixel. Therefore, combining the scans in such a way that preserves only the “good” regions will result in a complete fingerprint image thinned down to a one-pixel width. As a result, the remaining steps combine the two scans to preserve only the “good” regions from each scan.

- Step Three: removal of isolated noise

To begin with, the first step in removing the unwanted segments commences by performing another pixel-by-pixel scan across the entire image. A detected black pixel

is placed at the center of a seven-by-seven box, and the entire perimeter of this box is analyzed. At that point, the contents within the box are considered to be independent of all ridges in the fingerprint when all pixels located along the box perimeter are white. Therefore, the contents within the box are classified as “isolated noise” and are subsequently turned to white. Note that the contents within the box can not be classified as “isolated noise” but instead may belong to a larger ridge structure when at least one pixel along the perimeter is black. Figure 4.6 illustrates this difference by showing two different seven-by-seven boxes. Black pixels shown in the left box represent an isolated structure contained entirely within the box. Therefore, these pixels will be deleted. Note that one of the black pixels shown in the box on the right touches the perimeter. As a result, none of the pixel values in this box are altered.

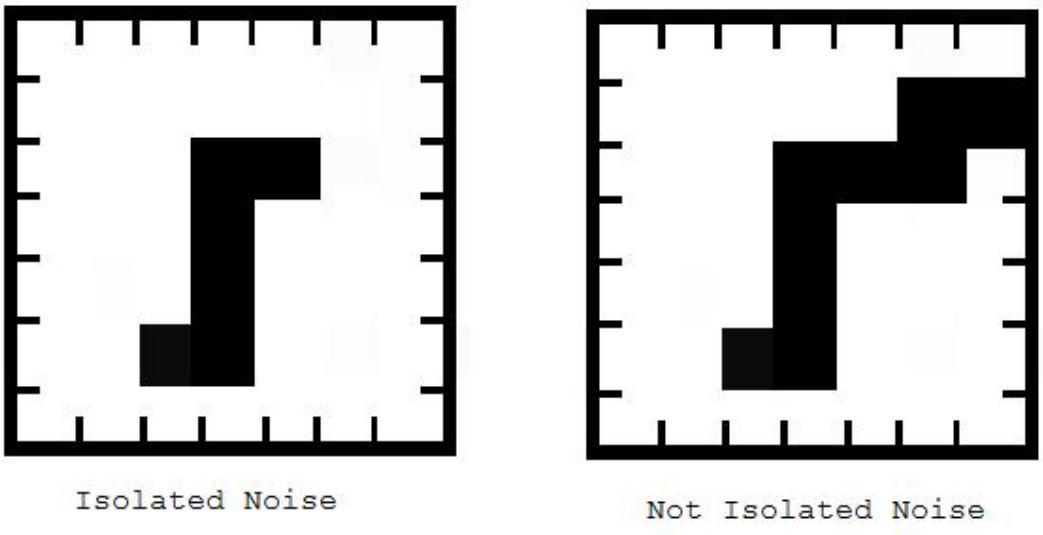


Figure 4.6. Seven-by-seven pixel box for removing isolated noise.

- Step Four: scan combination

Data from the two scans are added element by element after all potential “isolated noise” contributions have been removed from the two thinned images. The resulting combined matrix, however, now has values of zero, one, and two. A value of two means that the pixel from each scan was white, while a value of zero indicates the pixel from each scan was black. Meanwhile, a value of one means that the pixel from one scan was black while the same pixel from the other scan was white. As a result, the new matrix needs to be adjusted to represent a valid binary image containing only zeros and ones. Specifically, all zeros and ones are assigned a value of zero (black pixel), and all twos are assigned a value of one (white pixel). Note that values equal to one in the combined matrix are assigned a value of zero in the binary image because one of the scans produced a black pixel at these locations. This final adjustment can be accomplished by assigning a value of one to pixels with values found to be equal to two after this summation step, while all other values are transformed into zeros. In doing so, the resulting new matrix has been converted to zeros and ones and the two scans combined into one image. Combining the two scans contained in Figure 4.5 after removing the “isolated noise” from each image produces the image shown in Figure 4.7.

Figure 4.7. Combined image from both scans shown in Figure 4.5 following isolated noise removal

- Step Five: elimination of one pixel from two-by-two squares of black

Next, a new scan is conducted on the combined image to detect two-by-two blocks of black pixels which represent a location where a ridge has not been thinned to a one-pixel width. It is likely that some of these two-by-two blocks were created by the combination of the previous scans. This problem can be compensated for by changing one pixel within the block from black to white, which reduces the width at that particular point from two pixels to one. At the same time, this process needs to be implemented in a manner that preserves the overall ridge structure. This operation can be performed by analyzing the pixels touching each individual black pixel. Note that each black pixel touches the three other black pixels within the two-by-two

block. Therefore, there are only five other pixels that contain useful information.

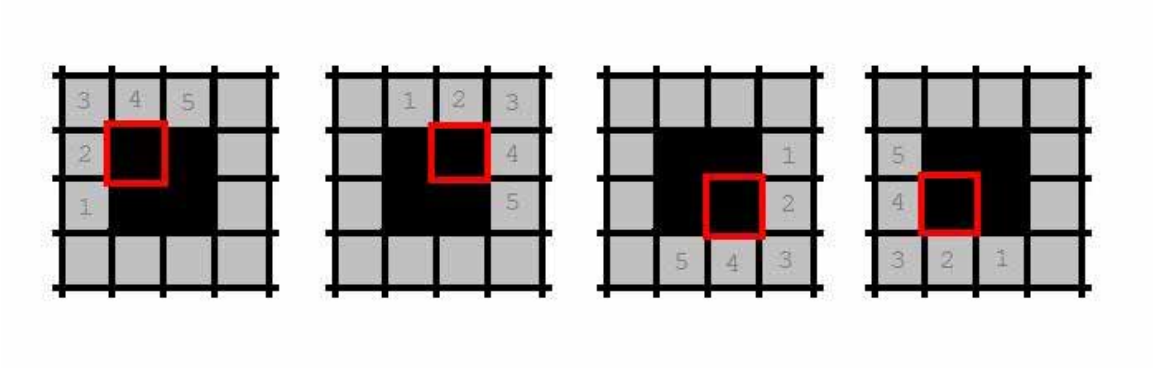


Figure 4.8. Surrounding pixels in two-by-two block.

In Figure 4.8, surrounding pixels to the two-by-two block are shown as gray because they may either be black or white. For each of the four pixels within the two-by-two block, the five pixels as outlined in Figure 4.8 are analyzed. In each case, the total number of black pixels in these five locations is recorded. At this point, the pixel with the least amount of black pixels surrounding it is turned from black to white. This pixel is deleted because it has the least amount of ridge information touching it, and its deletion will most likely preserve the overall ridge structure.

- Step Six: removal of unwanted spurs

At this point, the majority of the ridges have been reduced to a width of one-pixel. Looking at Figure 4.7, however, it should be evident that the overall ridge structure remains imperfect, due to the presence of short spurs jutting from several ridges. A magnified sample of these unwanted spurs is shown in Figure 4.9.

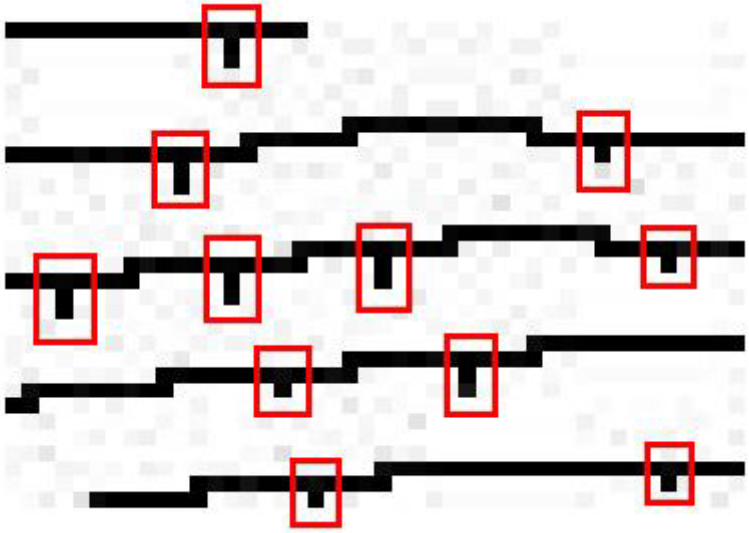


Figure 4.9. Unwanted spurs along ridges.

Upon observing these spurs, it becomes apparent that the end of the spur has the same characteristics as a ridge termination. Furthermore, note that the point where the spur connects to the ridge has the same qualities as a ridge bifurcation. This information is used in removing extraneous spurs. At this point, detecting terminations and bifurcations in a thinned image needs to be considered. One possible approach for this process involves computing what is referred to as the *crossing number* for each black pixel in the thinned image. As described in Maltoni, Maio, Jain, and Prabhakar (2003), the crossing number is defined as half the sum of differences between pairs of adjacent pixels that surround the given black pixel. In general terms, this computation begins by looking at the eight sets of adjacent pixels, as illustrated in Figure 4.10.

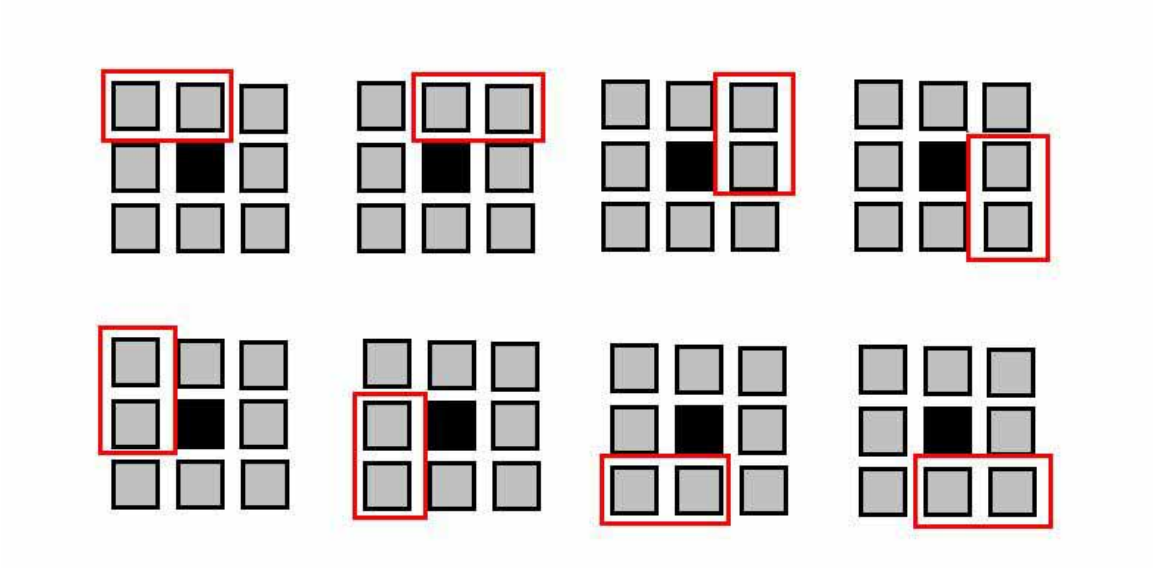


Figure 4.10. Eight sets of adjacent pixels used in computing crossing number.

Note that the difference between two adjacent pixels is equal to one when they are not of the same color. Conversely, this difference is zero when two pixels are of the same color. This difference is individually computed for the eight sets of adjacent pixels illustrated in Figure 4.10. Next, the eight differences are added together, and the resultant sum is divided by two. This value defines the crossing number for the black pixel at the center of the three-by-three pixel region. The center pixel corresponds to a termination minutia when the crossing number is equal to one. Similarly, the center pixel is the location of a bifurcation when the crossing number is greater than or equal to three, and it is an intermediate ridge point when the crossing number is equal to two. Figure 4.11 illustrates intra-ridge pixels, termination minutia, and bifurcation minutia as detected by the crossing number.

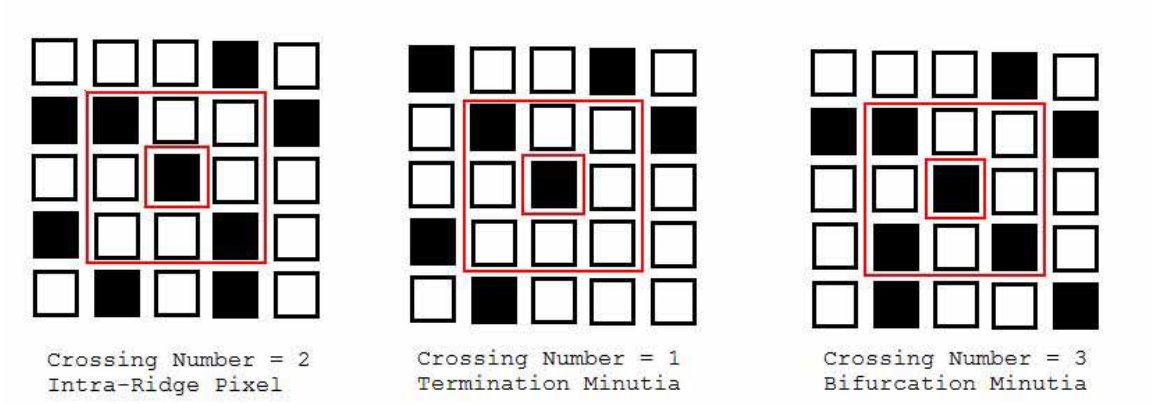


Figure 4.11. Visual examples of crossing number. (After: [Maltoni, Maio, Jain, & Prabhakar, 2003])

Therefore, the crossing number corresponding to each black pixel contained in the image is computed and used to delete unwanted spurs. First, the process cycles through all termination points and begins to trace the associated ridges. The ridge is determined to be an unwanted spur if the trace reaches a bifurcation point in less than twenty pixels. In such a case, the black pixels that have been traced from the bifurcation to the starting termination are turned to white, thus deleting the spur. On the other hand, if the trace length reaches twenty pixels before arriving at a bifurcation, the ridge is determined to be a valid ridge structure, and no change is made to the pixels. Figure 4.12 shows the result of applying this procedure.

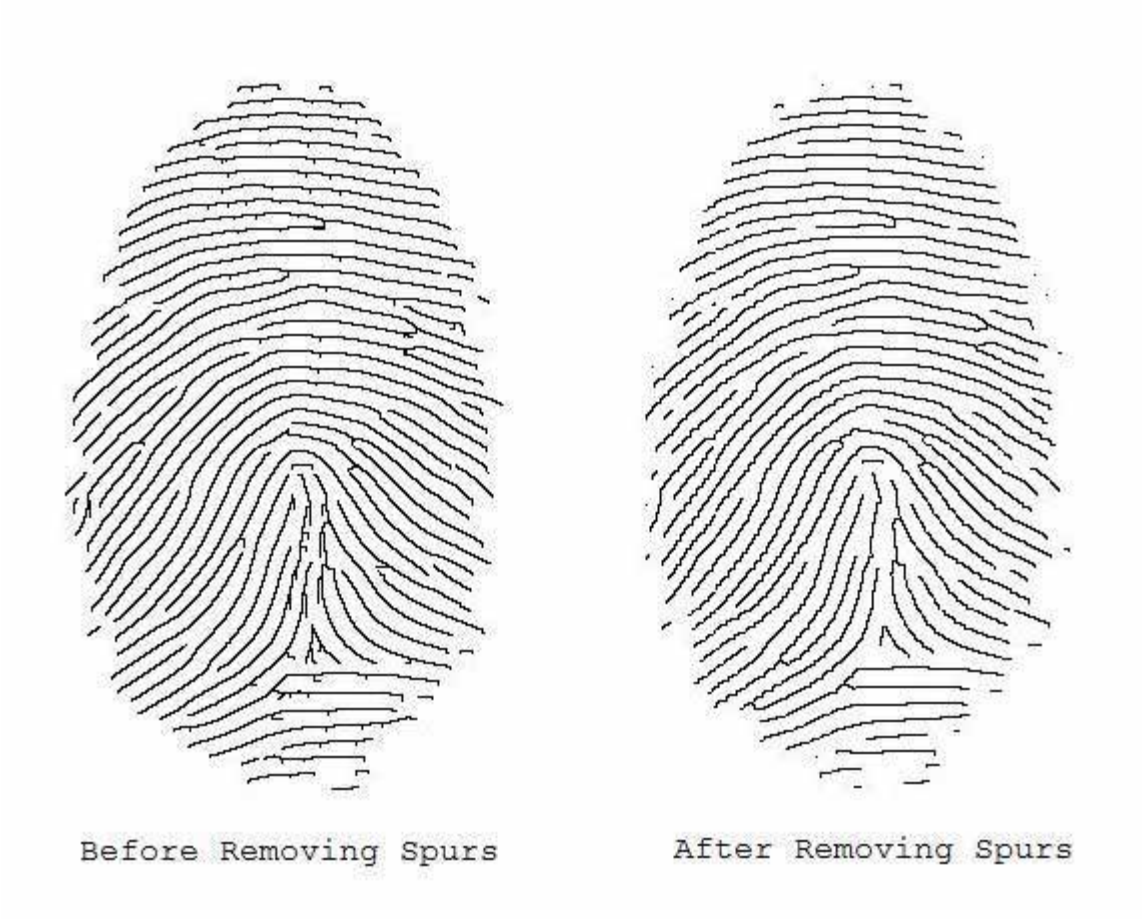


Figure 4.12. Impact of removing spurs.

- Step Seven: removal of duplicate horizontal and duplicate vertical lines

Observe that the thinned image has a better resemblance of the desired image after removing the unwanted spurs. Even so, there still may be a few problems that need to be taken into consideration, such as locations where a single ridge is represented by two horizontal lines or two vertical lines, as illustrated in Figure 4.13.

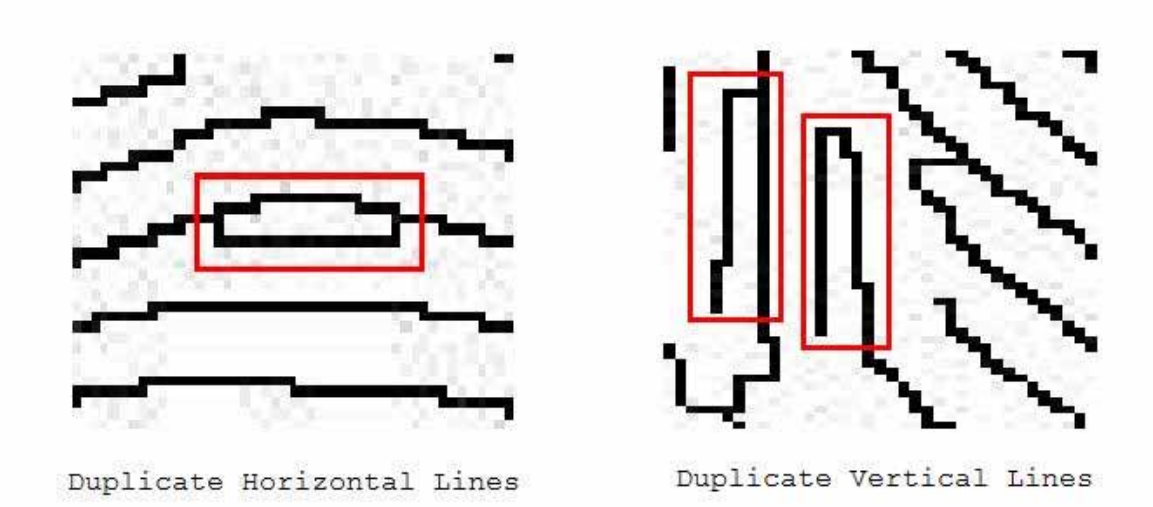


Figure 4.13. Duplicate horizontal and vertical lines.

Although these problems do not occur in every thinned image, they do occur enough to warrant attention. To remove the duplicate horizontal lines, the image is first scanned for horizontal lines with a length of five pixels. Upon locating one of these lines, the five rows immediately beneath this line are analyzed. If one of these rows also has a line with a length of five pixels in the same column indices as the initial line detected, then it is classified as a duplicate horizontal line. Therefore, the line of five pixels in the southernmost row is turned from black to white. Figure 4.14 displays this process.

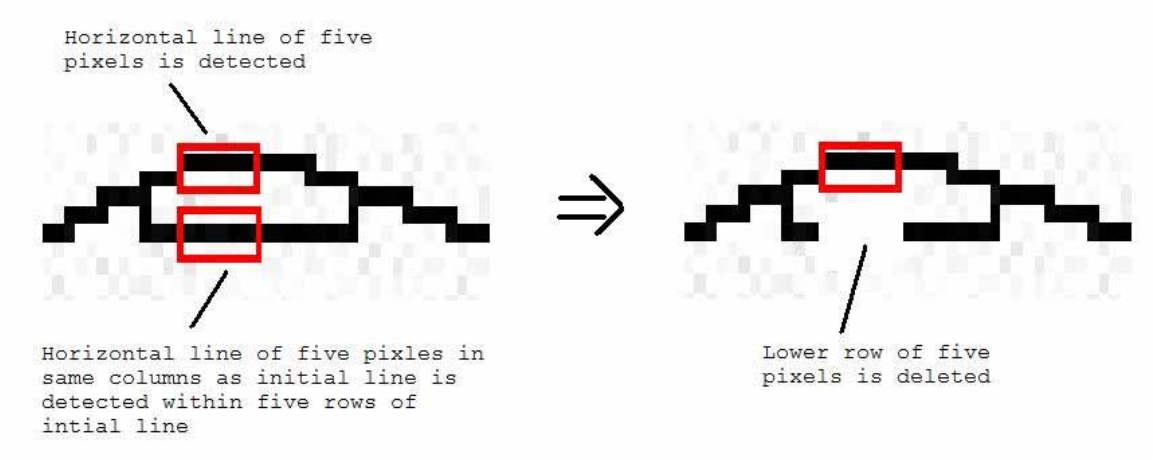


Figure 4.14. Deleting duplicate horizontal lines.

Once all duplicate horizontal lines have been accounted for, the resulting spurs can be deleted by applying the same process as before. A different procedure is necessary, however, to deal with duplicate vertical lines. Note that the duplicate vertical lines are longer segments that only connect in one place, while the duplicate horizontal lines are short segments that connect to the ridge in two places. For conceptual purposes, they can be treated as very lengthy spurs. Although they could be removed by simply increasing the maximum allowable trace length in the previous steps, doing so is not a recommended solution as it may result in unwillingly deleting a correct ridge that has a relatively short distance between a termination and bifurcation. This problem is addressed differently, as follows. The method begins by searching the thinned image for vertical lines with a length of ten pixels. When one such line is located, the same rows in the three columns to the right are analyzed. A duplicate vertical line exists when one of these columns also has a line with a length of ten pixels in the same rows. At this point, the two vertical lines are traced downward. The

trace continues until the ridge ends or a maximum trace length of forty pixels is reached, whichever comes first. As the traces are conducted, the total trace length is recorded for each segment. After the traces conclude, the total trace lengths for the two segments are compared. Since the vertical line that is part of the correct ridge structure does not come to a sudden end, it should reach the maximum trace length. Conversely, the vertical line that is the unwanted segment will reach an end before reaching the maximum trace length. Thus, the segment that has the shorter trace length is determined to be the unwanted spur and is subsequently deleted. Figure 4.15 displays this deletion process.

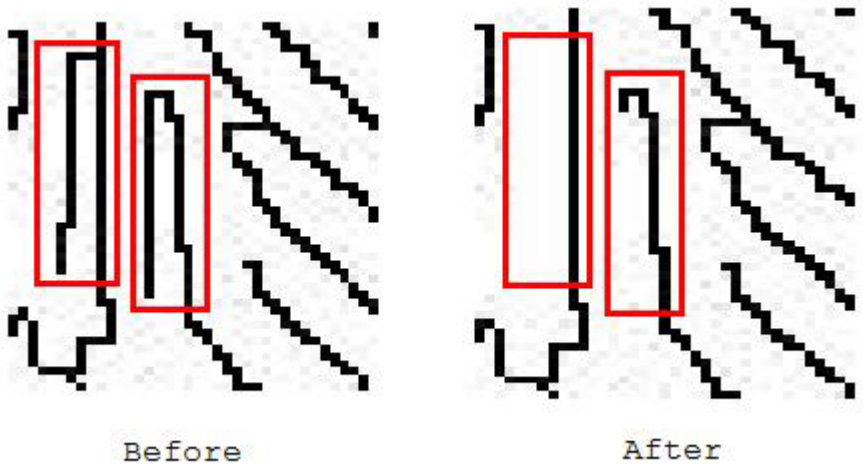


Figure 4.15. Deleting duplicate vertical lines.

All the steps in the block thinning method have been executed once this phase is complete. The final thinned image is shown in Figure 4.16.

Figure 4.16. Thinned image from block filtering.

Note that the block filtering does an effective job of thinning the majority of the original image even though there are a few imperfect areas within the thinned image. For example, some bifurcations in the original image appear as terminations in the thinned image.

b. Central Line

Central line thinning involves reducing the individual ridges to a width of one pixel at their central lines. The rule-based algorithm developed for character recognition by Ahmed and Ward (2002) can be applied to a fingerprint image. This scheme was studied because of its ability to effectively thin ridges. One significant advantage of this method is that it produces the same

thinned symbols regardless of rotation. This quality has specific appeal to a fingerprint recognition system, where it is likely the rotational orientation will be slightly different in successive login attempts. Additionally, it is stated that their proposed method preserves the topology and does not produce any discontinuity. These qualities are also important for an effective fingerprint thinning algorithm.

This thinning method is iterative in nature. At each iteration, the algorithm deletes those points that lie on the outer boundaries of the ridges, so long as the width of the ridge is greater than one pixel. The pixel will not be deleted if the process results in a disconnected graph [Ahmed & Ward, 2002]. The deletion process begins by scanning the image for black pixels. For each pixel, a check is performed to discover if it belongs to two pixels width in the vertical or horizontal directions. This step is necessary to ensure the pixels on the extremities of zigzag diagonal lines are not deleted. Figure 4.17 outlines this check for the vertical direction, while Figure 4.18 shows the check for the horizontal direction. The twenty-one rules mentioned at the end of these checks are used to determine which scenario, if any, applies to the associated three pixel-by-three pixel neighborhood. The first twenty rules were originally proposed by Ahmed and Ward (2002), while the twenty-first rule was introduced by Patil, Suralkar, and Sheikh (2005) for specific application to fingerprint images. This rule removes singular pixels, which are of interest in character

recognition but have no importance in fingerprint images. The twenty-one rules are shown in Figure 4.19 and will be discussed shortly.

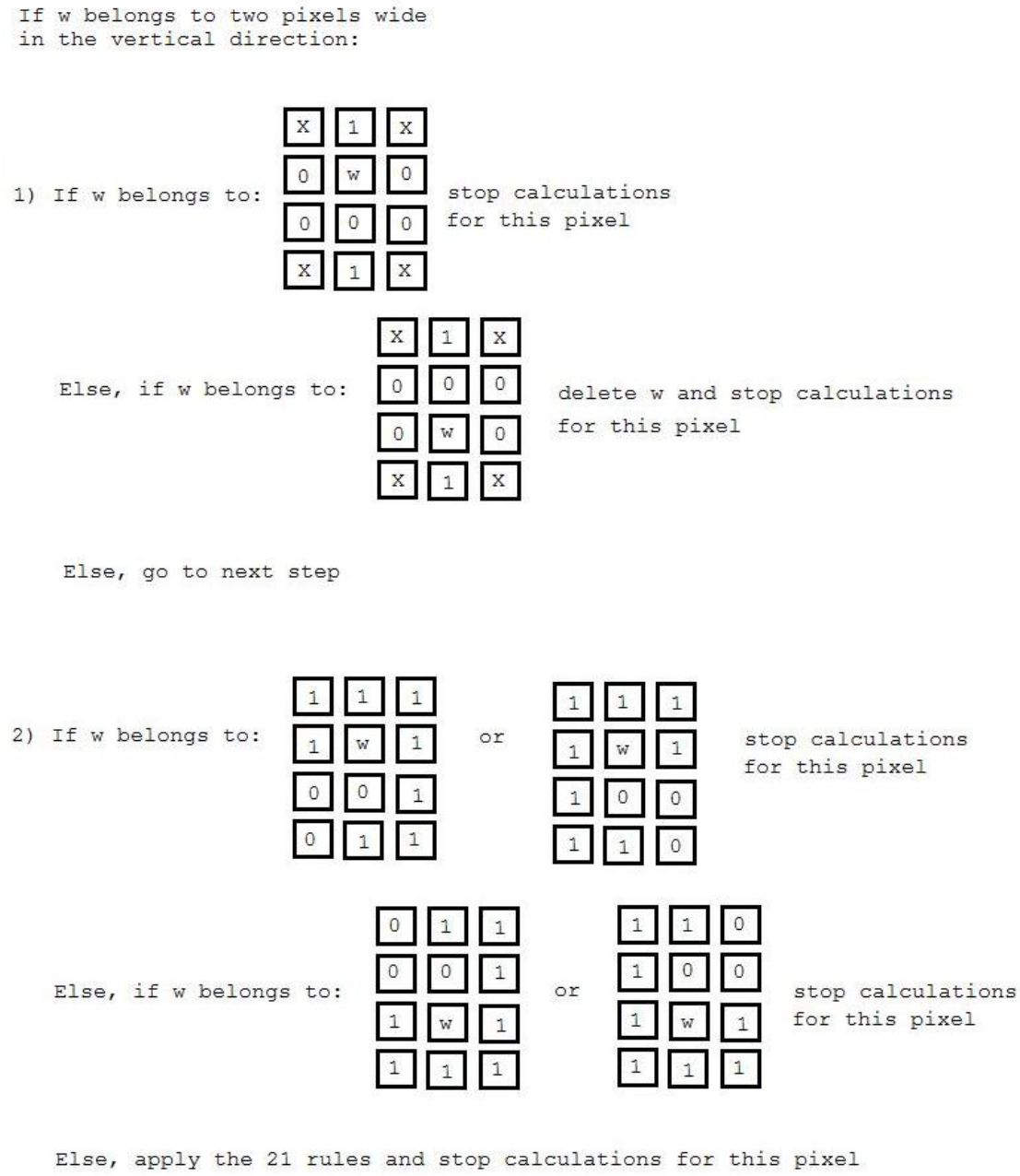


Figure 4.17. Two pixels width in vertical direction check. (After: [Ahmed & Ward, 2002])

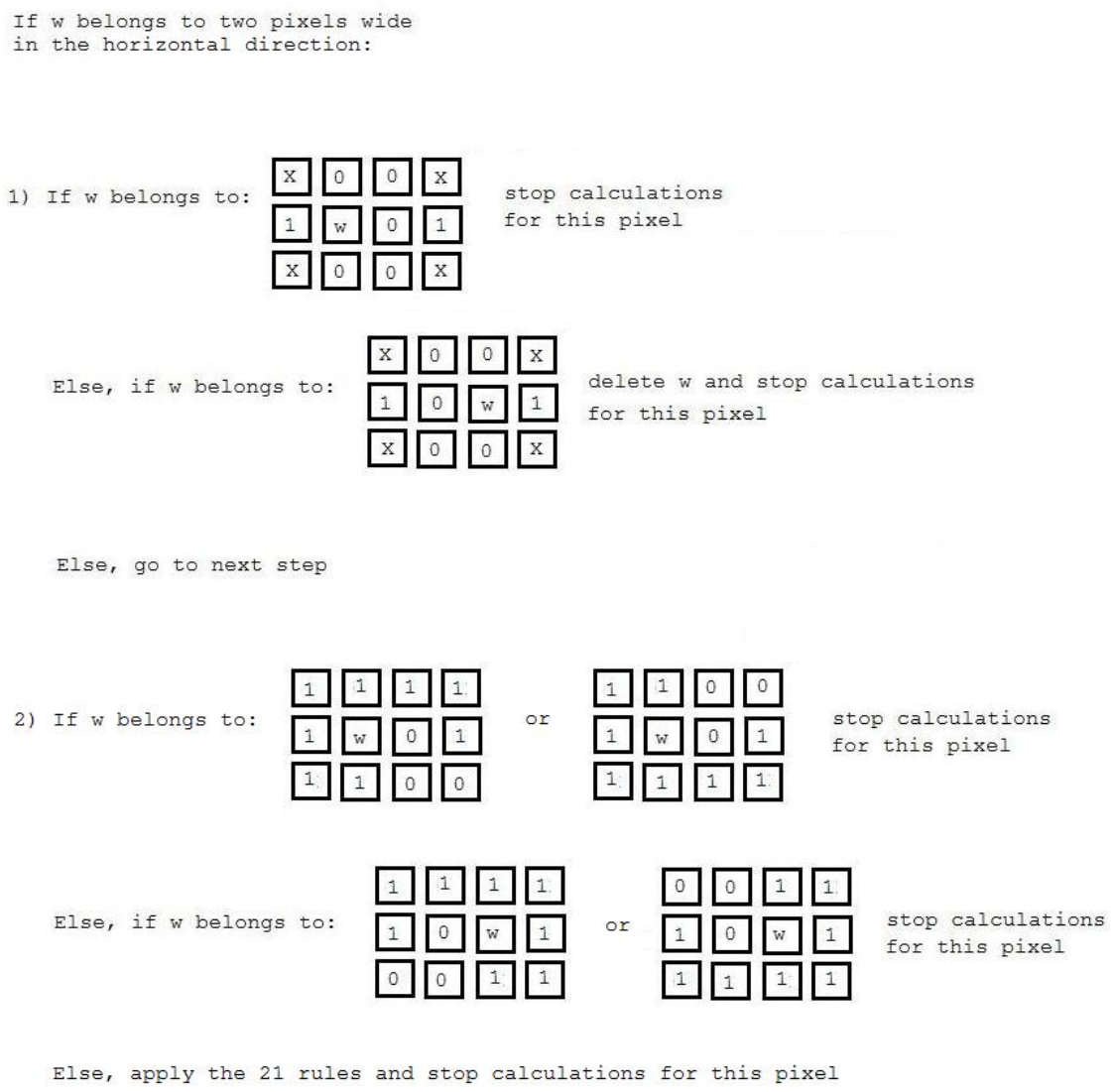


Figure 4.18. Two pixels width in horizontal direction check. (After: [Ahmed & Ward, 2002])

The twenty-one thinning rules are immediately applied if the pixel does not belong to a two pixels wide block in the vertical or horizontal direction. When a neighborhood satisfies one of the rules on the left side of the arrow, the middle pixel is turned to white, as indicated by the resulting neighborhood to the right of the arrow.

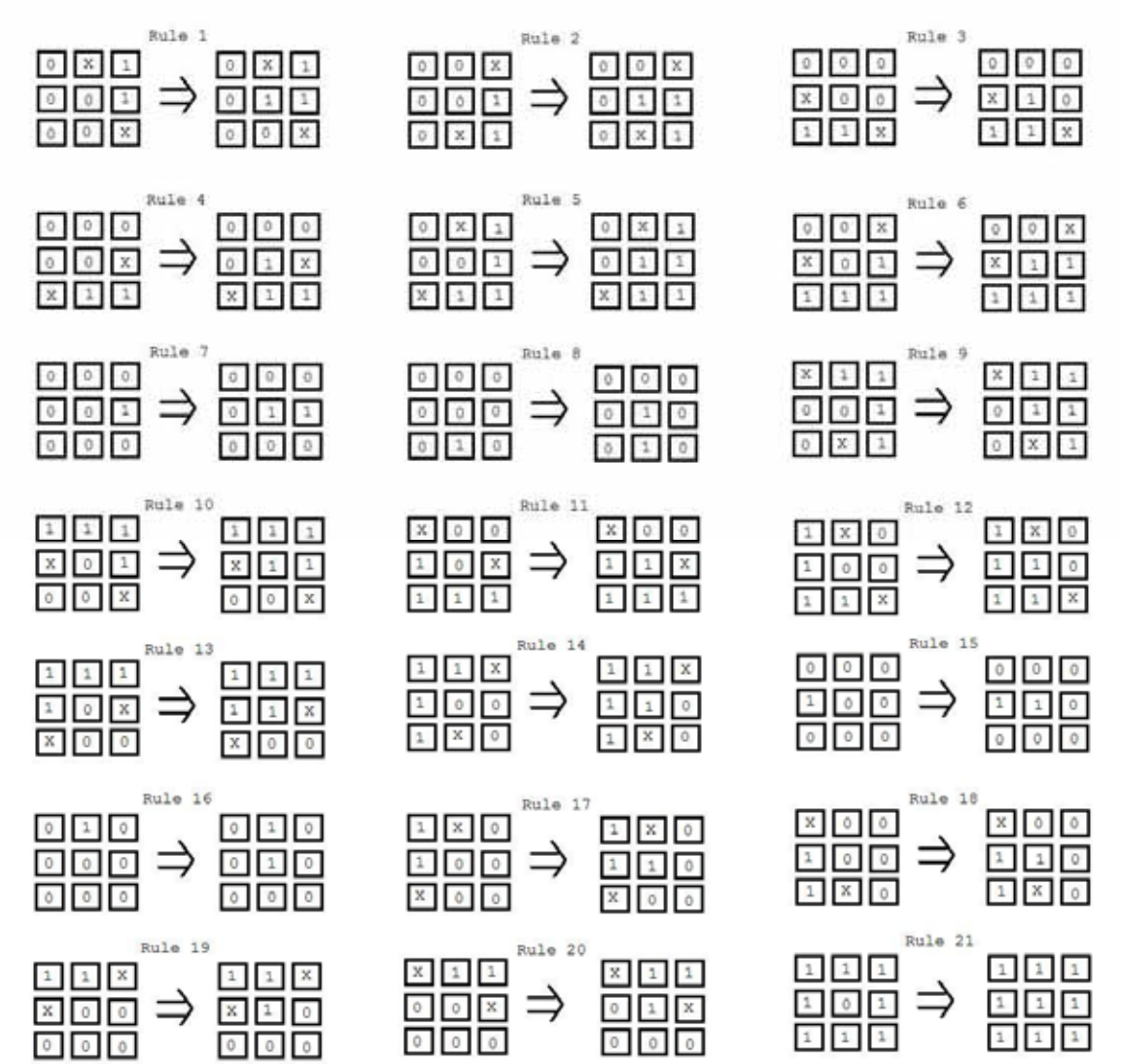


Figure 4.19. Thinning rules. (After: [Ahmed & Ward, 2002])

As stated previously, black pixels are denoted by a zero while white pixels are represented as a one. The X's are used to symbolize that the pixel can either be black or white. In other words, it does not matter what the value of these pixels are when the rules are applied. If none of the rules are satisfied, the middle pixel remains unchanged. The iteration continues, and the thinning progresses, until no changes occur from one

iteration to the next. Once the iteration is complete, Patil, Suralkar, and Sheikh (2005) discovered that the process does not completely thin diagonal lines to a width of one pixel. As a result, they proposed applying an additional set of rules to follow the thinning process proposed by Ahmed and Ward (2002), designed specifically for diagonal lines. In applying these rules, the continuity of the ridges does not get changed, and the original structure and angle of the ridges remains the same. The diagonal rules are depicted in Figure 4.20.

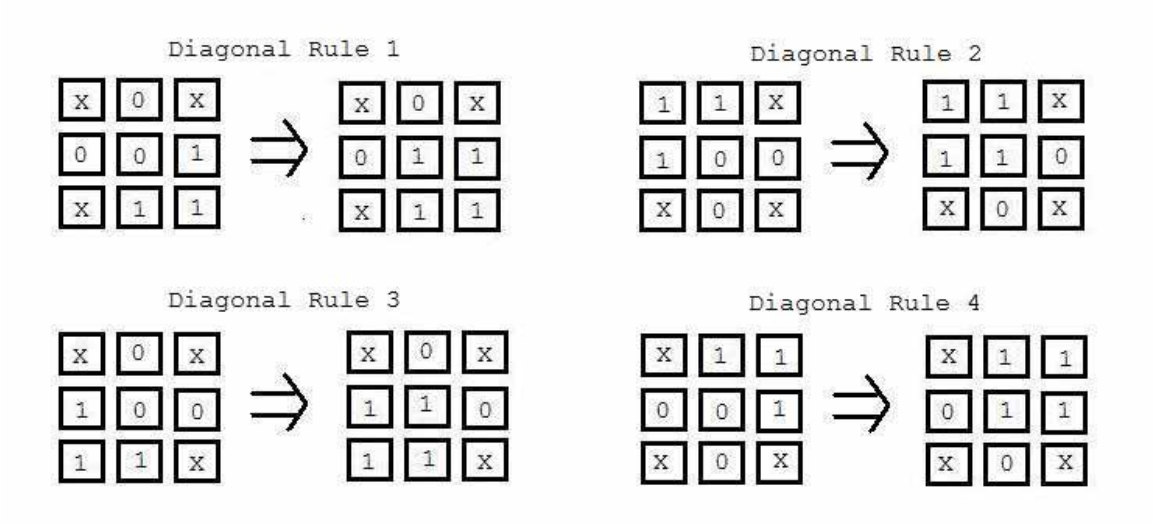


Figure 4.20. Diagonal thinning rules. (After: [Patil, Suralkar, & Sheikh, 2005])

After implementing the diagonal rules, the ridges of the fingerprint have been thinned to a one-pixel width at their central lines. Figure 4.21 displays the transformation of the black and white image to the final thinned image through the various iterations.

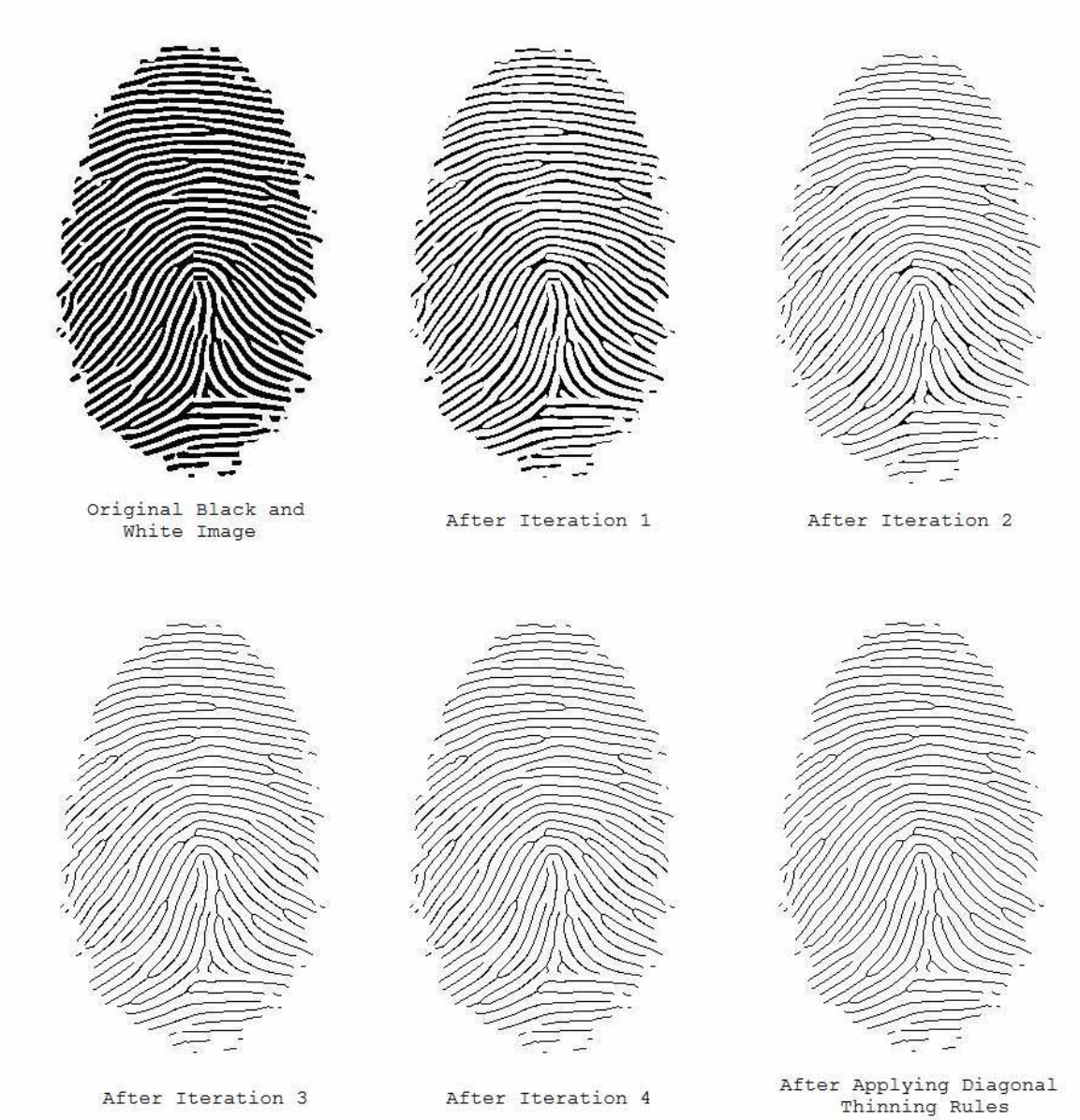


Figure 4.21. The thinning process to central lines.

In this visual example, the initial thinning is complete following the fourth iteration. The fifth iteration, however, is still conducted. Upon determining the fifth iteration produces no changes to the results from

the fourth iteration, the 21-rule thinning process stops, and the diagonal thinning rules are applied to produce the final thinned image.

3. Final Noise Removal

Following the thinning process, a final stage of noise removal is conducted to eliminate noise produced from the binarization and thinning processes. This stage focuses on removing the short island segments near the outer boundaries of the image which were produced during the binarization phase. Such segments need to be removed because they do not represent the true ridge structure of the original fingerprint. The process for deleting these segments is very similar to the process used in deleting the spurs produced from thinning by the block filter. It begins by detecting the terminations in the final thinned image. Starting at each termination, the corresponding ridge is traced one pixel at a time. Next, the segment is classified as a short island segment and is deleted if another termination is reached before the maximum trace length of seventeen pixels is reached. Conversely, the segment is said to be part of a complete ridge and not altered when the maximum trace length is reached without encountering another termination. Figure 4.22 shows the results of this process on an image that has been thinned by the central line technique.

Figure 4.22. Impact of deleting short island segments.

B. MINUTIAE EXTRACTION

The minutiae information can be extracted and stored after the image pre-processing is complete. This information consists of the following for each minutia:

- Location within the image
- Orientation angle
- Type (termination or bifurcation)

As described earlier, the crossing number is used again to locate the terminations and bifurcations within the final thinned image. In this process, the locations where the ridges end at the outer boundaries of the image are classified as terminations. In the true sense, however, these locations are not unique termination minutiae. Instead, they only appear as terminations because the dimensions of the image force each ridge to come to and end. Knowing this, these locations should not be recorded as minutiae within the fingerprint. One way to

eliminate such locations involves creating an ellipse to only select minutiae points inside the fingerprint image. The center of the ellipse is established by locating the minimum and maximum rows and columns that contain a ridge pixel, then calculating the row and column that lie halfway between these extremes. The major axis of this ellipse was empirically selected as 94% of the difference between the minimum and maximum rows containing a ridge pixel, whereas the minor axis was empirically selected as 86% of the difference between the minimum and maximum columns with a ridge pixel. Figure 4.23 shows the ellipse generated for a certain fingerprint.

Figure 4.23. Ellipse generated to reject ridge endings along the boundaries of an image.

Using this method, any termination or bifurcation that lies outside of the ellipse gets ignored. After this step is complete, the angles of the remaining minutiae are calculated. A termination angle is the angle between the horizontal and the direction of the ridge, while a bifurcation angle is the angle between the horizontal and the direction of the valley ending between the bifurcation. Figure 4.24 provides a visual description of these definitions.

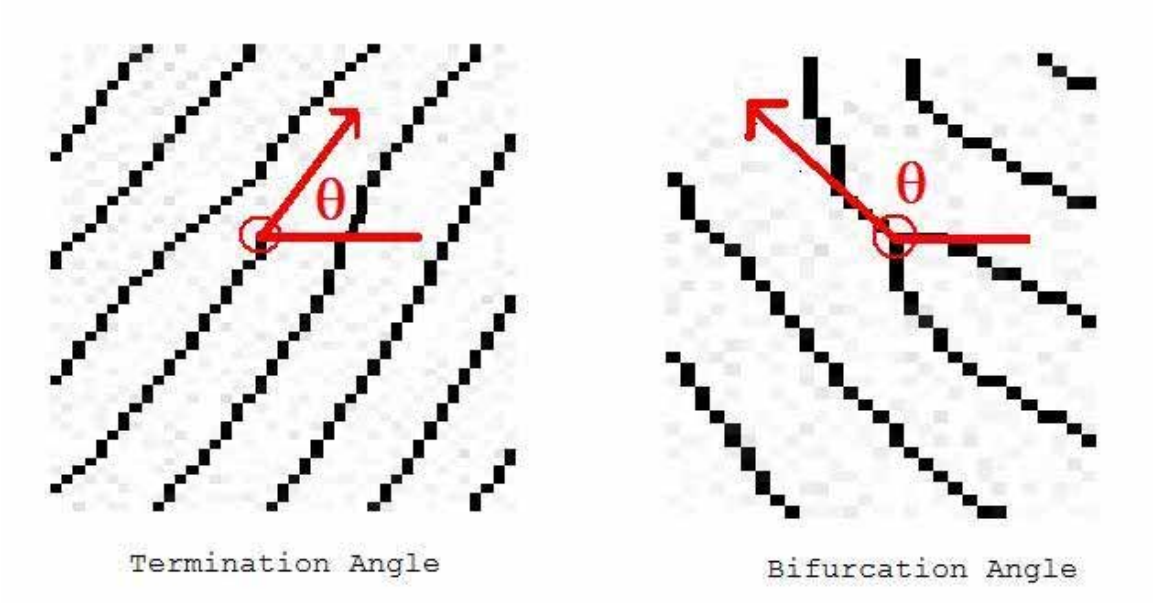


Figure 4.24. Definition of minutiae angles.

To compute the termination angles, the row and column indices for each termination are first recorded. Beginning at each termination, the corresponding ridge is traced backwards by five pixels, and the resulting row and column indices are stored. Care must be taken to ensure the angle is calculated correctly. Figure 4.25 summarizes the rules developed by the author in calculating each angle.

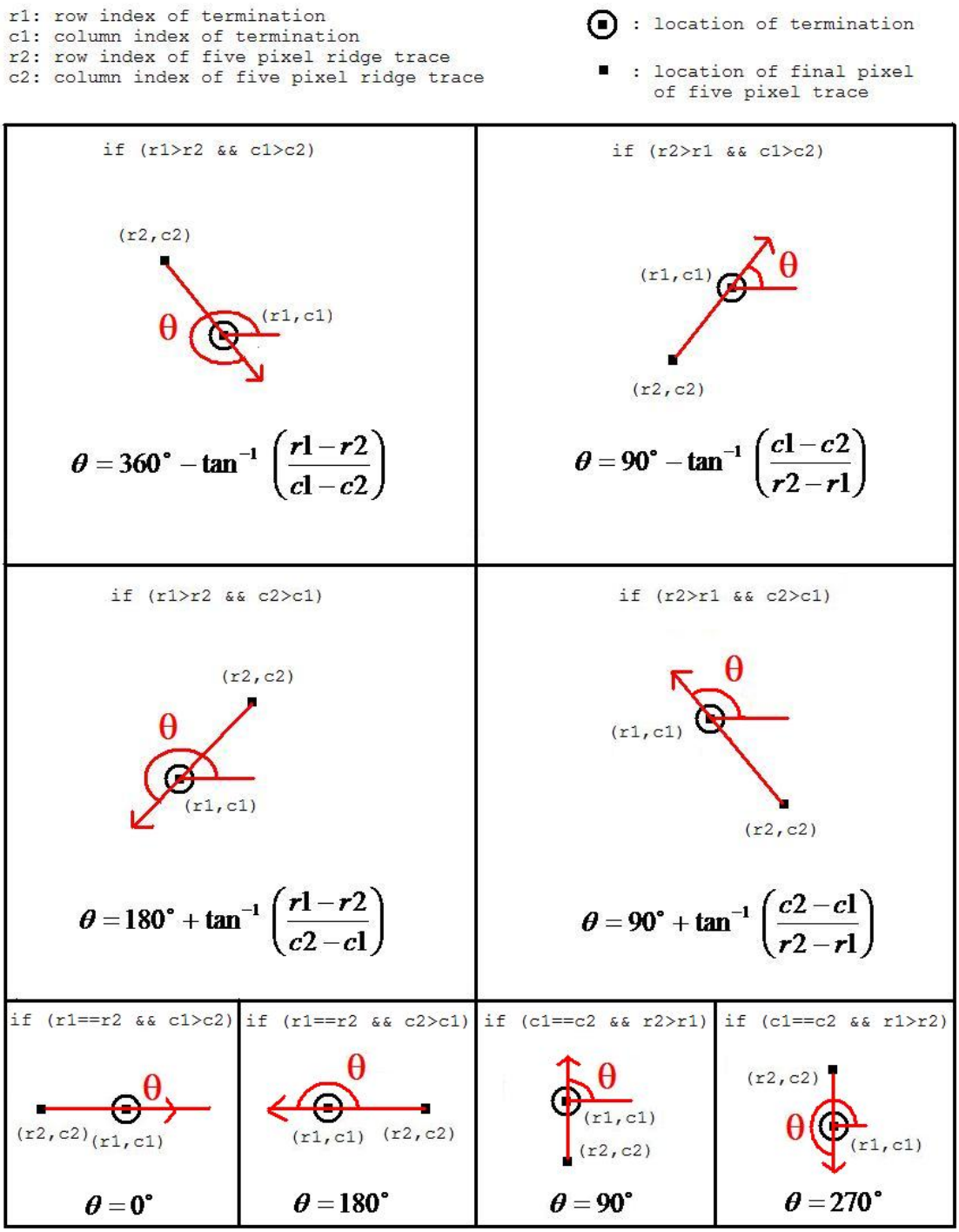


Figure 4.25. Rules for calculating termination angles.

These rules allow for 360° coverage while using the inverse tangent function. The process for computing the bifurcation angles uses the same set of rules in a slightly

different manner. In particular, it takes advantage of what is known as termination/bifurcation duality [Maltoni, Maio, Jain, & Prabhakar, 2003]. This property says that a termination in a black and white image corresponds to a bifurcation in its negative image. Similarly, a bifurcation in the black and white image is in the same position as a termination in the negative image. Figure 4.26 shows this relationship.

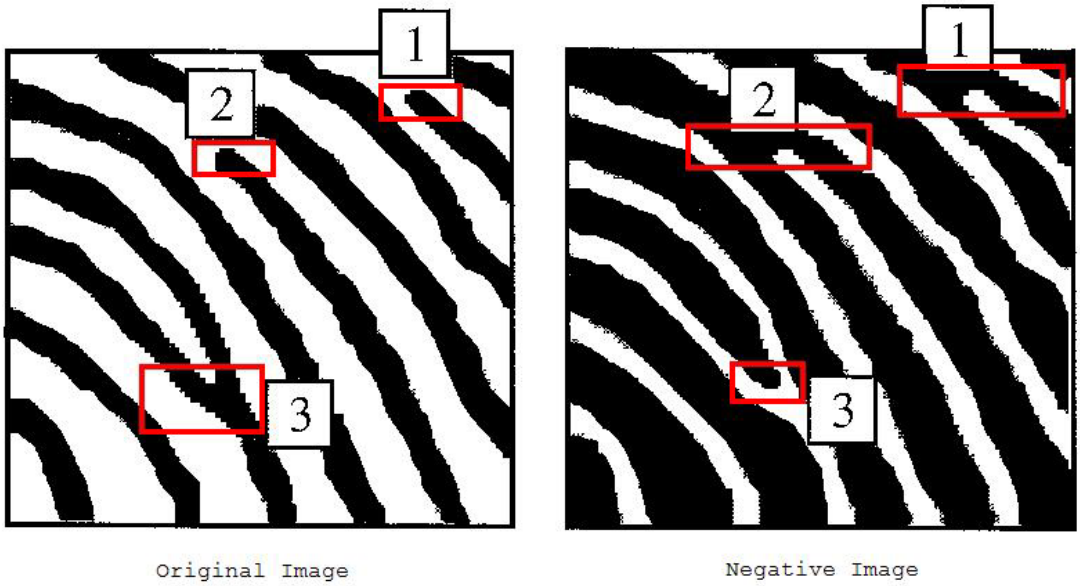


Figure 4.26. Termination/bifurcation duality. (After: [Maltoni, Maio, Jain, & Prabhakar, 2003])

As a result, the original black and white image is inverted to obtain the bifurcation angles. The negative image is then thinned using the methods described previously. At this point, the terminations in the thinned image are detected using the crossing number once again. The angle of each termination is then computed by applying the rules shown in Figure 4.25. From here, positions of the bifurcations in the original image are compared to the

positions of the terminations in the inverted image. Each bifurcation angle is determined by locating the termination in the inverted image that lies nearest the position of the bifurcation. The angle of this termination is then assigned as the angle of the corresponding bifurcation. Note that the angle is assigned an arbitrary angle of 0° when no terminations in the inverted image lie within a distance of 100 pixels of a bifurcation. This step was added to ensure an angle is assigned to every bifurcation, regardless of unforeseen errors.

A final step is applied in an effort to remove unwanted termination minutiae along the outer boundaries of the image. If the input fingerprint is slightly rotated, some terminations may lie within the previously created ellipse. In such a case, these minutiae will not be deleted, and further processing is applied to remove them as these points are not true ridge terminations. Note that terminations residing within a fingerprint are surrounded by other ridges, while terminations along the outer boundaries are not completely surrounded. Thus, the angles of the terminations are used to conduct a search for nearby ridges. A horizontal search is performed when the angle is between 45° and 135° or between 225° and 315°. Here, the twenty columns to the immediate right and left of the termination are examined. If a ridge is not detected in one of the directions, the termination is deleted because it lies on the boundary of the image. In a similar manner, a vertical search is conducted when the termination angle is between 135° and 225° or between 315° and 45°. This search looks at the twenty rows immediately above and below the termination. If either direction does not locate a

ridge, the termination under examination is deleted. Otherwise, the termination is not modified when the search locates a ridge in both directions.

Following this step, the position and angle of the final set of minutiae is known. For visual purposes, this data is plotted to observe the detection ability of the various thinning methods. In the plot, terminations are denoted by a square while bifurcations are displayed as diamonds. Also, the angle of each minutia is shown by a short line segment jutting out of the squares and diamonds. Figure 4.27 displays the minutiae position and orientation produced by block filter thinning and central line thinning. Figure 4.28 shows the minutiae data from block filter thinning superimposed on the original fingerprint image, and Figure 4.29 displays the same for the central line thinning results.

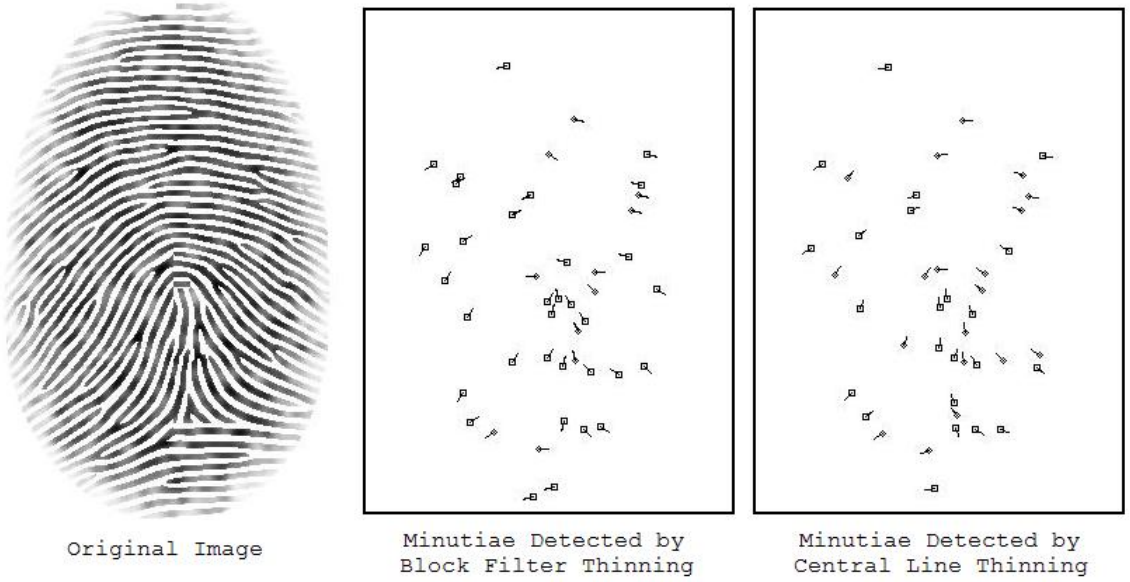


Figure 4.27. Display of minutiae detected by block filter and central line thinning.

Figure 4.28. Magnified display of minutiae detected by block filter thinning superimposed on original fingerprint. Terminations denoted by a square, bifurcations denoted by a diamond.

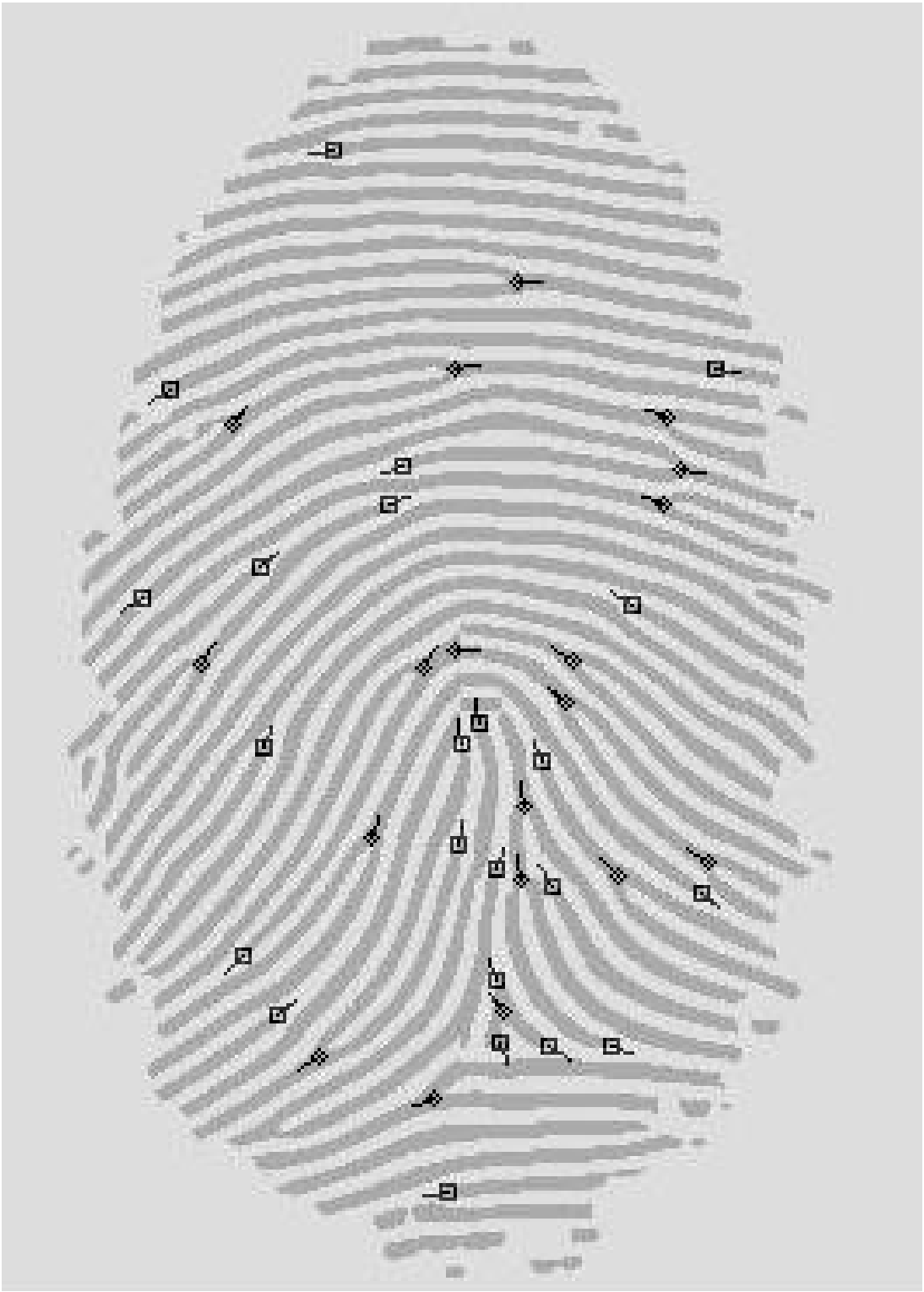


Figure 4.29. Magnified display of minutiae detected by central line thinning superimposed on original fingerprint. Terminations denoted by a square, bifurcations denoted by a diamond.

Both thinning methods appear to do an adequate job at locating the minutiae. The difference in detected minutiae shown in Figure 4.28 and 4.29 arises simply because the two thinning methods produce slightly different results. An error that occurs in the thinning process leads to the minutiae extraction process detecting minutiae that do not correspond with the original ridge structure. Although the block filtering produces some spurious minutiae, the majority of the detected minutiae are accurate and near the correct location. Meanwhile, the central line thinning does an excellent job of maintaining the ridge integrity of the original image. The performance of each technique will be examined in the coming chapters.

C. CONCLUSION

Image pre-processing is the most critical step for reliable minutiae detection. Accurate estimation of the overall ridge structure in the final thinned image is an essential step needed to ensure accurate estimation of the fingerprint minutiae. Errors in the thinning process result in errors in the minutiae extraction process, making the fingerprint identification step essentially worthless when such errors occur. The following chapter discusses minutiae data matching issues.

V. MINUTIAE MATCHING

The next essential step in the fingerprint recognition process is the comparison of minutiae data. Much like the minutiae extraction steps, reliable matching is necessary for an effective system. This chapter discusses a matching process used in comparing minutiae data.

A. DATA FORMAT

First, the data extracted in the previous steps must be stored in a consistent format to ensure proper comparisons are conducted. In the approach considered in this thesis, minutiae data from a single fingerprint is stored in a matrix format, where the number of rows represents the number of minutiae points. The number of columns in this matrix is fixed at a value of four. The first column indicates the row index of each minutia, and the second column indicates the column index of the minutiae in the fingerprint image. The third column provides the angle orientation of the minutiae, and the fourth column indicates the type of minutiae, where values of one or two are used to indicate a termination or bifurcation, respectively. Table 5.1 provides a sample of this matrix of data extracted for each fingerprint.

Row Index	Column Index	Orientation Angle	Type of Minutiae
312	170	329.04	1
266	171	120.96	1
312	188	338.2	1
185	194	149.04	1
268	214	321.34	1
117	218	348.69	1
202	70	59.036	2
133	79	51.34	2
315	104	210.96	2
252	119	68.199	2

Table 5.1. Sample matrix of minutiae data.

B. MATCHING PROCESS

The matching process involves comparing one set of minutiae data to another set. In most cases, this process compares an input data set to a previously stored data set with a known identity, referred to as a template. The template is created during the enrollment process, when a user presents a finger for the system to collect the data from. This information is then stored as the defining characteristics for that particular user.

The method introduced in the following steps is based on the matching process presented by Luo, Tian, and Wu (2000). This process begins by creating a matrix, called *rotatevalues* in the code included in Appendix A, of the orientation angle difference between each template minutiae, T_k $(1 \le k \le NT)$, and each input minutiae, I_m $(1 \le m \le NI)$. Here, NT and NI represent the total number of minutiae in the template and input sets, respectively. The value at *rotatevalues(k,m)* represents the difference between the orientation angles of T_k and I_m. T_k and I_m represent the extracted data in all the columns of row k and row m in the template and input matrices, respectively.

Next, template and input minutiae are selected as reference points for their respective data sets. This process is performed for each possible combination of template and input minutiae points. It begins by selecting the first minutiae point in the template matrix, T_1, along with the first minutiae point in the input matrix, I_1. Once the matching procedure for these reference points is complete, it performs the same process using T_1 and I_2 as reference points. After all combinations of T_k and T_m have been used as reference points, the matching process concludes.

In each case, the reference points are used to convert the remaining data points to polar coordinates. Equation 5.1 shows the procedure followed for converting the template minutiae from row and column indices to polar coordinates:

$$\begin{pmatrix} r_k^T \\ \phi_k^T \\ \theta_k^T \end{pmatrix} = \begin{pmatrix} \sqrt{\left(row_k^T - row_{ref}^T\right)^2 + \left(col_k^T - col_{ref}^T\right)^2} \\ \tan^{-1}\left(\dfrac{row_k^T - row_{ref}^T}{col_k^T - col_{ref}^T}\right) \\ \theta_k^T - \theta_{ref}^T \end{pmatrix}. \qquad (5.1)$$

Converting to polar coordinates allows for an effective match process to be conducted regardless of any rotational or translational displacement between the template and input images. The variable r_k^T represents the radial distance, ϕ_k^T represents the radial angle, and θ_k^T represents the orientation of the k^{th} minutia, all with

respect to the reference minutiae. Also, the variables row_k^T and col_k^T refer to the row and column indices of the k^{th} minutia in the template matrix, while row_{ref}^T and col_{ref}^T refer to the indices of the reference minutia currently being used for the template matrix. Equation 5.2 provides the polar coordinate transformation for the input minutiae:

$$\begin{pmatrix} r_m^I \\ \phi_m^I \\ \theta_m^I \end{pmatrix} = \begin{pmatrix} \sqrt{\left(row_m^I - row_{ref}^I\right)^2 + \left(col_m^I - col_{ref}^I\right)^2} \\ \tan^{-1}\left(\dfrac{row_m^I - row_{ref}^I}{col_m^I - col_{ref}^I}\right) + rotatevalues(k,m) \\ \theta_m^I - \theta_{ref}^I \end{pmatrix}. \qquad (5.2)$$

Instead of comparing row and column indices, the comparison is now based on the relative position of the minutiae with regards to the reference minutiae. The relative positions of the minutiae remain the same even as the absolute position of the fingerprint image changes. As a result, converting the data to polar coordinates in the matching process creates a more robust system since the alignment of an input image is likely to be different than the alignment of the template image.

Following the conversion to polar coordinates, the resulting template and input data sets are sent through a matching process, which compares the polar data of each input minutiae to each template minutiae. For each possible combination, the difference between the template and input data is computed for the radial distance, radial angle, and orientation angle. Two minutiae are determined to match if all of the following criteria are met:

- The absolute radial distance difference is less than a tolerance of three pixels.
- The absolute radial angle difference is less than a tolerance of five degrees.
- The absolute orientation angle difference is less than a tolerance of ten degrees.
- The two minutiae are the same type.

Throughout this process, a counter keeps track of the total number of minutiae that are matched. Once a minutia in the template is matched to a minutia in the input, its type is assigned a value of three to prevent it from being matched to more than one minutia. Next, a matching score representing the similarity between the datasets is calculated for each combination of reference points using the total number of minutiae matched in each case. Equation 5.3 shows the formula for calculating the matching score when T_k and I_m are being used as reference minutiae:

$$matchingscore(k,m) = \frac{\#\,\text{matching minutiae}}{\max(\text{NT}, \text{NI})}, \qquad (5.3)$$

where NT and NI represent the total number of minutiae in the template and input matrices, respectively. The denominator is assigned the maximum value between NT and NI. By this definition, the matching score takes on a value between zero and one. A matching score of one indicates the data matches perfectly, whereas a matching score of zero occurs when there are no matching minutiae. The denominator is defined in this manner to thwart efforts of an imposter attempting to gain access by preventing someone from simply creating a fraudulent input consisting of an incredibly large number of minutiae in hopes of

randomly matching only a few of them. Since the number of matching minutiae can never exceed the minimum value between *NT* and *NI*, dividing this number by a large *NI* number would result in a low matching score, even if a lot of minutiae matches are formed.

Matching scores have been calculated for every possible combination of reference points and stored in the *matchingscore* matrix. The final matching score between the template and input datasets is selected as the maximum value in the *matchingscore* matrix. Note that this maximum value represents the best possible alignment between the two sets of data, and the input is considered to come from the same finger as that represented in the template image when the matching score is greater than a previously user-defined threshold value. The input image is found to come from a different fingerprint than that represented in the current template when the matching score is below the threshold value.

C. CONCLUSION

The polar coordinate system discussed in this section allows matching to be performed regardless of the template and input image orientation. This characteristic is a significant advantage because it allows for slight variations in finger positions during successive recognition attempts as the input image does not have to be aligned exactly as it was during the enrollment process. The following chapter presents experimental results of the complete fingerprint recognition system.

VI. EXPERIMENTAL RESULTS

This chapter presents the experimental procedures used in testing the developed fingerprint recognition system. It also provides the results obtained from the various tests and discusses the thinning method that produces the best results.

A. SYNTHETIC DATABASE GENERATION

As stated previously, the SFINGE software was used to develop fingerprints free of sensor noise. A total of sixty base images were created for the database. These images covered the five major classes of fingerprints outlined in Figure 3.2. Additionally, the width of the ridges was not held constant. This allowed the database to more accurately represent a sample of fingerprints from the general population. In order to conduct an efficient test, a file naming convention was developed for the images. The base images were assigned a name in the format ‘s00##_1.jpg’, where ## represents the image number ranging from 00 to 59.

Next, a series of transformations was applied to generate new images of the same fingerprint using these base images. These transformations included horizontal shifts to the left and right, vertical shifts up and down, and clockwise and counterclockwise rotations. The horizontal and vertical shifts were on the order of forty to eighty pixels, and the rotations ranged from five to fifteen degrees. Table 6.1 outlines the transformations and their corresponding filename convention.

Transformation	filename
Original Image	s00##_1.jpg'
Shift Left	s00##_2.jpg'
Shift Right	s00##_3.jpg'
Shift Up	s00##_4.jpg'
Shift Down	s00##_5.jpg'
Clockwise Rotation	s00##_6.jpg'
Counterclockwise Rotation	s00##_7.jpg'

Table 6.1. Filename convention for transformations.

All transformations were applied to each original image. Starting with sixty original images, this process produced seven total impressions for each fingerprint. Thus, the final database consisted of 420 synthetically generated fingerprints.

B. TEMPLATE CREATION

Using the original images, two templates were created for each of the sixty fingerprints. This was performed by implementing all the steps outlined in Chapter IV on each fingerprint image. Since the fingerprints were sent through the minutiae extraction process, the templates were simply a matrix of data organized in the same format as that shown in Table 5.1. One set of templates used the central line thinning method and was saved using the filename 's00##_t.mat', while the other templates were generated using the block filter thinning method and given the name 's00##_t2.mat'. By creating separate templates for the two thinning processes, a performance evaluation between the two methods could be conducted.

C. SIMULATION

Simulations were conducted to determine numerical values of the False Non-match Rate (FNMR) and False Match

Rate (FMR) for various thresholds. Equation 6.1 expresses the formula used for calculating the FNMR at different threshold values:

$$FNMR = \frac{\#\,\text{False Non - matches}}{\#\,\text{Enrollee Attempts}}\,. \qquad (6.1)$$

The experiment matched the seven images of the same fingerprint to the corresponding template for the same fingerprint. Therefore, each match that took place was considered to be an enrollee attempt to access the system. With seven impressions of sixty fingerprints, there were a total of 420 enrollee attempts. A False Non-match was recorded when the matching score between an enrollee and its template was less than the established threshold. The procedure for calculating the FMR was slightly different. Equation 6.2 presents the formula used in calculating the FMR:

$$FMR = \frac{\#\,\text{False Matches}}{\#\,\text{Imposter Attempts}}\,. \qquad (6.2)$$

The imposter attempts were implemented by matching the seven images of one fingerprint with all the templates from the 59 other fingerprints. This procedure simulated an imposter attempt as the input images did not have a corresponding template in the database. Without a template, the imposter should not be allowed access. This process was applied for the seven impressions from all 60 different fingerprints. In each case, the seven impressions were matched with the other 59 templates, resulting in 7 x 59 x 60 = 24780 imposter attempts. A false match was recorded for each imposter attempt when the matching score was greater than the established threshold. The final FMR was computed by dividing the total number of False Matches by the total number of imposter attempts.

Figure 6.1 summarizes the matching process for computing the FNMR, while Figure 6.2 shows the process for the FMR.

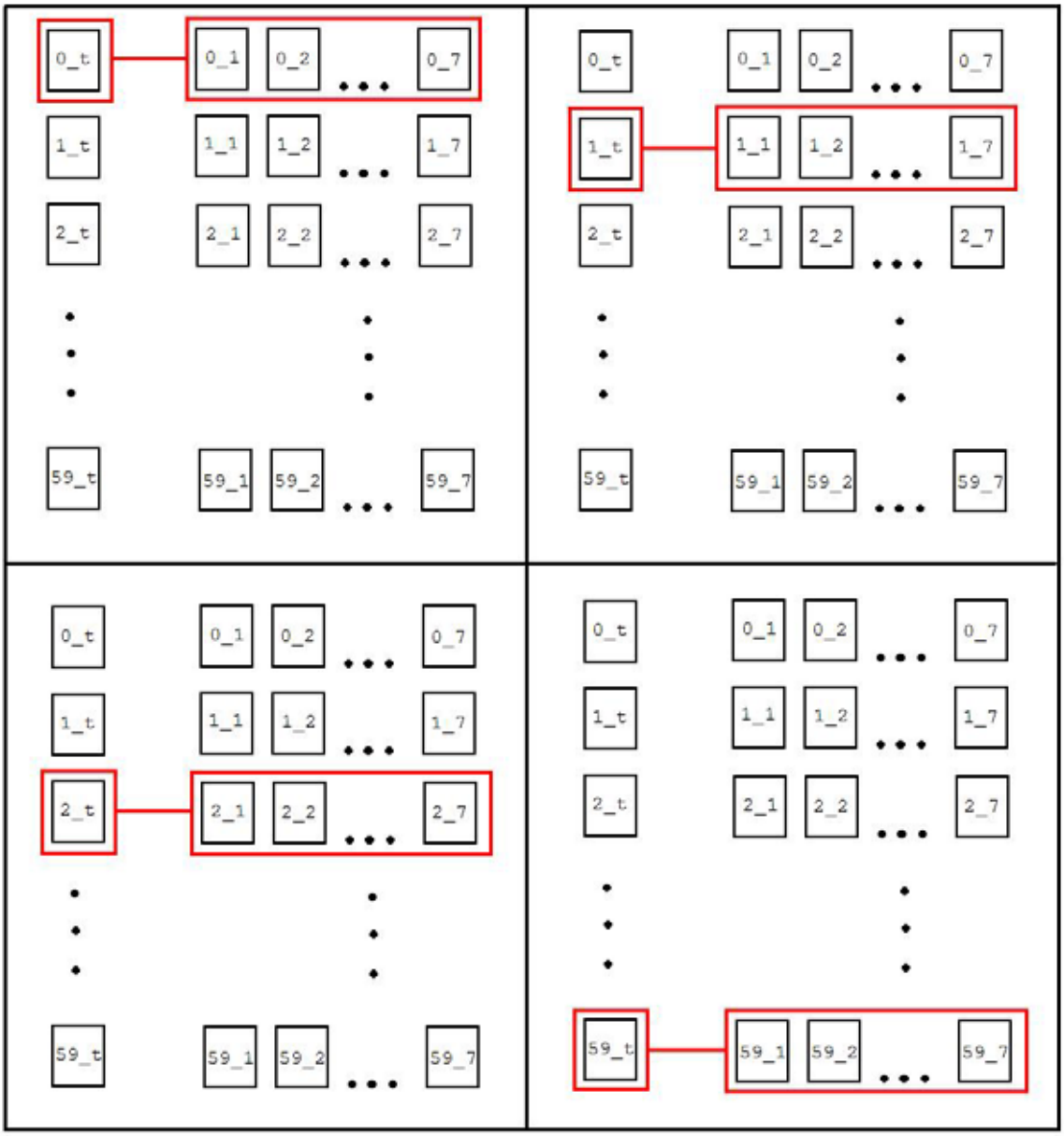
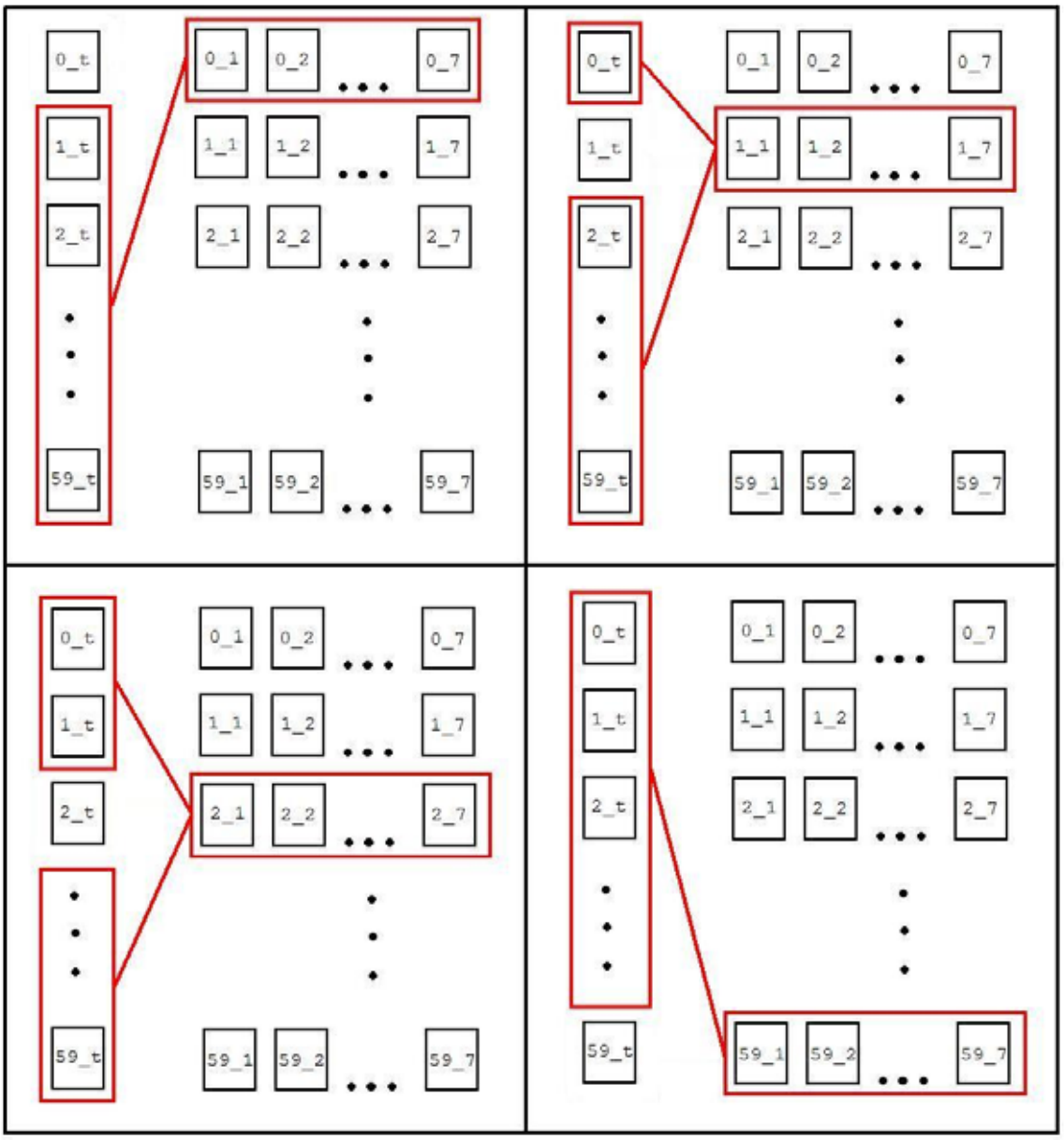


Figure 6.1. Matching process for computing the FNMR.



The seven impressions from each fingerprint are compared to the 59 templates from the other fingerprints to generate the False Match Rate

Figure 6.2. Matching process for computing the FMR.

FNMR and FMR matching processes were performed for the two methods of thinning. In each case, the matching scores were compared against various threshold values to obtain the performance data for each thinning method. The threshold was adjusted from 0.05 to 0.95 in increments of

0.05. Table 6.2 shows the results obtained for the central line thinning method.

Threshold Value	# FNM	Enrollee Attempts	FNMR		# FM	Imposter Attempts	FMR
0.95	301	420	0.7167		0	24780	0.0000
0.90	241	420	0.5738		0	24780	0.0000
0.85	191	420	0.4548		0	24780	0.0000
0.80	136	420	0.3238		0	24780	0.0000
0.75	106	420	0.2524		0	24780	0.0000
0.70	67	420	0.1595		0	24780	0.0000
0.65	40	420	0.0952		0	24780	0.0000
0.60	15	420	0.0357		0	24780	0.0000
0.55	3	420	0.0071		0	24780	0.0000
0.50	0	420	0.0000		0	24780	0.0000
0.45	0	420	0.0000		0	24780	0.0000
0.40	0	420	0.0000		0	24780	0.0000
0.35	0	420	0.0000		0	24780	0.0000
0.30	0	420	0.0000		0	24780	0.0000
0.25	0	420	0.0000		0	24780	0.0000
0.20	0	420	0.0000		0	24780	0.0000
0.15	0	420	0.0000		3	24780	0.0001
0.10	0	420	0.0000		320	24780	0.0129
0.05	0	420	0.0000		13088	24780	0.5282

Table 6.2. Central line thinning FNMR/FMR data.

Results show that the central line thinning scheme leads to very good recognition rates for threshold values below 0.65, and that the FNMR begins to increase to unacceptable values when the threshold is greater than 0.65. For example, at a threshold of 0.75, the FNMR is 0.2542, meaning that an accepted enrollee will be denied access to the system 25.42% of the time. Although the same user may be granted access on his next attempt, this threshold may create too much user inconvenience in most applications. Meanwhile, the FMR only reaches unacceptable values at threshold levels below 0.15. Thus, a system threshold of 0.55 would be adequate in most applications,

leading to FNMR and FMR below 1% and essentially 0%, respectively. A more secure application may require increasing the threshold to around 0.65 to further reduce the possibility of a False Match occurring. Next, Table 6.3 provides the experimental results obtained with the block filter thinning technique.

Threshold Value	# FNM	Enrollee Attempts	FNMR		# FM	Imposter Attempts	FMR
0.95	315	420	0.7500		0	24780	0.0000
0.90	260	420	0.6190		0	24780	0.0000
0.85	203	420	0.4833		0	24780	0.0000
0.80	167	420	0.3976		0	24780	0.0000
0.75	142	420	0.3381		0	24780	0.0000
0.70	130	420	0.3095		0	24780	0.0000
0.65	125	420	0.2976		0	24780	0.0000
0.60	123	420	0.2929		0	24780	0.0000
0.55	120	420	0.2857		0	24780	0.0000
0.50	120	420	0.2857		0	24780	0.0000
0.45	120	420	0.2857		0	24780	0.0000
0.40	120	420	0.2857		0	24780	0.0000
0.35	120	420	0.2857		0	24780	0.0000
0.30	120	420	0.2857		0	24780	0.0000
0.25	117	420	0.2786		0	24780	0.0000
0.20	106	420	0.2524		0	24780	0.0000
0.15	86	420	0.2048		13	24780	0.0005
0.10	45	420	0.1071		277	24780	0.0112
0.05	4	420	0.0095		12092	24780	0.4880

Table 6.3. Block filter thinning FNMR/FMR data.

Results show the block filter thinning method develops problems in recognizing authorized enrollees. Note that the FNMR does not reach 10% until the threshold has been reduced to 0.10. At this threshold, the FMR has a value of 1%. Thus, there does not appear to be an acceptable threshold for a system that provides both user convenience and security from unauthorized personnel. Upon further examination, the False Non-matches that occur for

thresholds below 0.55 were produced solely by the rotated input images. For horizontal and vertical displacements, the system was still able to match authorized users. Therefore, another set of data was computed for thinning by block filter. This time, the input images with a rotational transformation were not used. Instead, only the base image (‘s00##_1.jpg’), the image shifted left (‘s00##_2.jpg’), the image shifted right (‘s00##_3.jpg’), the image shifted up (‘s00##_4.jpg’), and the image shifted down (‘s00##_5.jpg’) were used in the matching process. This procedure produced the data given in Table 6.4.

Threshold Value	# FNM	Enrollee Attempts	FNMR		# FM	Imposter Attempts	FMR
0.95	195	300	0.6500		0	17700	0.0000
0.90	140	300	0.4667		0	17700	0.0000
0.85	83	300	0.2767		0	17700	0.0000
0.80	47	300	0.1567		0	17700	0.0000
0.75	22	300	0.0733		0	17700	0.0000
0.70	10	300	0.0333		0	17700	0.0000
0.65	5	300	0.0167		0	17700	0.0000
0.60	3	300	0.0100		0	17700	0.0000
0.55	0	300	0.0000		0	17700	0.0000
0.50	0	300	0.0000		0	17700	0.0000
0.45	0	300	0.0000		0	17700	0.0000
0.40	0	300	0.0000		0	17700	0.0000
0.35	0	300	0.0000		0	17700	0.0000
0.30	0	300	0.0000		0	17700	0.0000
0.25	0	300	0.0000		0	17700	0.0000
0.20	0	300	0.0000		0	17700	0.0000
0.15	0	300	0.0000		12	17700	0.0007
0.10	0	300	0.0000		251	17700	0.0142
0.05	0	300	0.0000		9834	17700	0.5556

Table 6.4. Block filter thinning FNMR/FMR data using no rotated input images.

As Table 6.4 confirms, the block filter thinning technique produces good data when the input image is not

rotated. When this is the case, a threshold around 0.55 would be effective for most applications. Even so, limiting input images to no rotation is difficult to employ in practice. Therefore, the central line thinning process is more appealing for a practical application as it is rotational invariant.

D. CONCLUSION

Results show that using the central line thinning method leads to good recognition rates and allows the recognition system to handle rotated images without performance degradation. The central line thinning method's main strength lies in its ability to thin ridges in the same manner regardless of rotational orientation, making it easier to effectively process all types of input images. Results also show that the block filter thinning works well for horizontal and vertical displacements, but degrades significantly when dealing with rotated images. As a result, the central line thinning method is the better choice for thinning the ridges to a one-pixel width.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

Many steps are required for a reliable fingerprint recognition system. The first step involves pre-processing the original fingerprint image. This includes image binarization, ridge thinning, and noise removal. After this pre-processing phase is complete, the minutiae extraction step is executed, where the minutiae data is collected and organized into a single matrix. From here, the matrix of input data is matched with the template data to determine if the input comes from the same finger as one of the templates. Since all steps are necessary for an effective system, no one step is more important than another. Instead, all steps should be viewed as critical steps for a reliable and effective system.

B. RECOMMENDATIONS

Possible recommendations for future work include developing code to handle fingerprints containing a large amount of noise. This extension would be necessary to apply this system to commonly collected fingerprints, as most fingerprint images captured from a sensor contain some degree of sensor noise. In such a case, an additional noise removal step needs to be applied before the image is sent to the binarization step. This extension project would require focusing on transforming a noisy fingerprint image to a black and white image while preserving the overall ridge structure. Providing this de-noising phase can be implemented successfully, the remaining steps of the fingerprint recognition system can then be applied as they were in this research study. Furthermore, a more advanced

area of future work could involve incorporating this system with another biometric in developing a multimodal system.

APPENDIX A. MATLAB CODE

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **maincode.m**

```
%-----------------------------------------------------------------------------------------------
% When this code is executed, the data for FNMR and FMR is
% generated for the two thinning methods
% -----------------------------------------------------------------------------------------------
% Output: Matrices of data containing matching scores
%
% Functions Used: get_minutiae_data.m, minutiae_match.m
%
% Author: Graig Diefenderfer
% Date: May 2006
%-----------------------------------------------------------------------------------------------

clear
falseNM_matchscores=zeros(60,7);
falseNM_matchscores2=zeros(60,7);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute False NonMatches via central thinning

tic
for kt=0:59
   kt

   if kt < 10  %put zero in front of single character numbers
      a1=['s000',num2str(kt),'_t.mat'];
   else
      a1=['s00',num2str(kt),'_t.mat'];
   end

   a1t=load(a1);
   at=a1t.templatedata;

   for ki=1:7

      a2=[a1(1:6),num2str(ki),'.bmp'];   %a1(1:6)=s00##_
      ai=get_minutiae_data(a2,1);

      input_data=ai;
```

```
        template_data=at;

        ismatch=minutiae_match(input_data,template_data);
        falseNM_matchscores(kt+1,ki)=ismatch;   %kt+1 because kt starts counting at zero

    end
end
toc     %Elapsed time is 2521.245433 seconds

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute False NonMatches via boundary block thinning

tic
for kt=0:59
    kt

    if kt < 10  %put zero in front of single character numbers
        a1=['s000',num2str(kt),'_t2.mat'];
    else
        a1=['s00',num2str(kt),'_t2.mat'];
    end

    a1t=load(a1);
    at=a1t.templatedata;

    for ki=1:7

        a2=[a1(1:6),num2str(ki),'.bmp'];   %a1(1:6)=s00##_
        ai=get_minutiae_data(a2,2);

        input_data=ai;
        template_data=at;

        ismatch=minutiae_match(input_data,template_data);
        falseNM_matchscores2(kt+1,ki)=ismatch;   %kt+1 because kt starts counting at zero

    end
end
toc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute False Matches via central thinning
tic
falseM_matchscores=zeros(60,59,7);
```

```
for ka=0:59
   ka
   if ka < 10  %put zero in front of single character numbers
      a1=['s000',num2str(ka),'_'];
   else
      a1=['s00',num2str(ka),'_'];
   end

   for kb=1:7

      a1i=[a1,num2str(kb),'.bmp'];
      ai=get_minutiae_data(a1i,1);
      input_data=ai;
      comparecounter=0;

      for kc=0:59 %cycle through templates

         if kc < 10  %put zero in front of single character numbers
            a1t=['s000',num2str(kc),'_t.mat'];
         else
            a1t=['s00',num2str(kc),'_t.mat'];
         end

         if kc~=ka   %don't match input to its correct template
            comparecounter=comparecounter+1;
            a1td=load(a1t);
            at=a1td.templatedata;
            template_data=at;

            ismatch=minutiae_match(input_data,template_data);
            falseM_matchscores(ka+1,comparecounter,kb)=ismatch;
         end
      end

   end
end

toc


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute False Matches via boundary block thinning
tic
falseM_matchscores2=zeros(60,59,7);

for ka=0:59
```

```
    ka
    if ka < 10  %put zero in front of single character numbers
       a1=['s000',num2str(ka),'_'];
    else
       a1=['s00',num2str(ka),'_'];
    end

    for kb=1:7

       a1i=[a1,num2str(kb),'.bmp'];
       ai=get_minutiae_data(a1i,2);
       input_data=ai;
       comparecounter=0;

       for kc=0:59 %cycle through templates

          if kc < 10  %put zero in front of single character numbers
             a1t=['s000',num2str(kc),'_t2.mat'];
          else
             a1t=['s00',num2str(kc),'_t2.mat'];
          end

          if kc~=ka   %don't match input to its correct template
             comparecounter=comparecounter+1;
             a1td=load(a1t);
             at=a1td.templatedata;
             template_data=at;

             ismatch=minutiae_match(input_data,template_data);
             falseM_matchscores2(ka+1,comparecounter,kb)=ismatch;
          end
       end

    end
end

toc

save('falseNMdata.mat','falseNM_matchscores')
save('falseNMdata2.mat','falseNM_matchscores2')
save('falseMdata.mat','falseM_matchscores')
save('falseMdata2.mat','falseM_matchscores2')
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **thin_blockfilter.m**

```
function bwthin=thin_blockfilter(xbw)

%----------------------------------------------------------------------------------------------------
% This is the main code for thinning ridges via the block filter method
% ---------------------------------------------------------------------------------------------------
% Input: Black and White input image in matrix format
%
% Output: Thinned image
%
% Functions Used: removeblock4.m, detect_term_bif.m, linetrace.m
%                        removehorizduplicate.m, removevertduplicate.m
%
% Author: Graig Diefenderfer
% Date: May 2006
%----------------------------------------------------------------------------------------------------


%This function attempts to thin the black and white input image to a
%1-pixel width

[row,col]=size(xbw);

%%%%%figure(1),imshow(xbw)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% erode original image

g=[1,1;1,1];
z=imdilate(xbw,g);
%%%%%figure(2),imshow(z)
xbw=z;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%put border of white around image
xbw(1:5,:)=1;
xbw(row-4:row,:)=1;
xbw(:,1:5)=1;
xbw(:,col-4:col)=1;

a=xbw;
b=xbw;
```

```
% Left to Right
for k=1:row
   for m=1:col

      if a(k,m)==0       %Black pixel
            a(k+1:k+3,m+1:m+3)=1;
      end

   end
end

a=a(1:row,1:col);       %Ensure a is the same size as original image

%Right to Left
for k=1:row
   for m=col:-1:1

      if b(k,m)==0

         if m>4
            b(k+1:k+3,m-3:m-1)=1;
         end

      end

   end
end

b=b(1:row,1:col);

%%%%%figure(4),imshow(a)
%%%%%figure(5),imshow(b)
%%%%%figure(9),imshow((a+b)/2)

d=((a+b)/2);
d=im2bw(d,0.95);
d=d-0;

%remove isolated noise

for k=1+3:row-3
   for m=1+3:col-3

      %if entire perimeter of 7x7 box is white, the contents within the box are unwanted
      %noise
```

```
      if a(k,m)==0
         if sum(a(k-3:k+3,m-3))==7 && sum(a(k-3:k+3,m+3))==7 && …
                  sum(a(k-3,m-3:m+3))==7 && sum(a(k+3,m-3:m+3))==7

            a(k-3:k+3,m-3:m+3)=1;
         end
      end

      if b(k,m)==0
         if sum(b(k-3:k+3,m-3))==7 && sum(b(k-3:k+3,m+3))==7 && …
                  sum(b(k-3,m-3:m+3))==7 && sum(b(k+3,m-3:m+3))==7

            b(k-3:k+3,m-3:m+3)=1;
         end
      end

   end
end

%%%%%figure(6),imshow(a)
%%%%%figure(7),imshow(b)

c=(a+b)/2;
c=im2bw(c,0.95);
c=c-0;
%%%%%figure(8),imshow(c)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% scan image for 2x2 blocks of black, then remove one pixel from block

c2=removeblock4(d);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% detect end of lines and bifurcations using crossing number

[lineends,bifur]=detect_term_bif(c2);

%%%%%figure(10),imshow(lineends)
%%%%%figure(13),imshow(bifur)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% remove unwanted short line segments

f1=linetrace(c2,lineends,bifur,20);
figure(14),imshow(f1)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Remove duplicate horizontal lines

f2=removehorizduplicate(f1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Remove duplicate vertical lines

f2a=removevertduplicate(f2);
%%%%%figure(15),imshow(f2a)

[lineends2,bifur2]=detect_term_bif(f2a);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% remove unwanted line segments left from duplicate horizontal lines

f3=linetrace(f2a,lineends2,bifur2,20);
%%%%%figure(16),imshow(f3)

bwthin=f3;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **removeblock4.m**

```
function y=removeblock4(x)

%------------------------------------------------------------------------------------------------
% This function deletes one pixel from a two-by-two square of black
% pixels after initial block filter thinning process
% ------------------------------------------------------------------------------------------------
% Input: Initial thinned image, in matrix form
%
% Output: Thinned image, free of two-by-two squares of black pixels
%
% Author: Graig Diefenderfer
% Date: May 2006
%------------------------------------------------------------------------------------------------

[row,col]=size(x);

for k=3:row-2
   for m=3:col-2
```

```
        blocksum=x(k,m)+x(k,m+1)+x(k+1,m+1)+x(k+1,m);

        %blocksum = 0 when there is a square of 4 black pixels

        if blocksum == 0

           % 1 2
           % 4 3

           %compute number of black pixels adjacent to outside of each
           %pixel in block

           touch1= 5 - sum(x(k-1:k+1,m-1))+sum(x(k-1,m:m+1));
           touch2= 5 - sum(x(k-1,m:m+2))+sum(x(k:k+1,m+2));
           touch3= 5 - sum(x(k:k+2,m+2))+sum(x(k+2,m:m+1));
           touch4= 5 - sum(x(k:k+2,m-1))+sum(x(k+2,m:m+1));

           touchvec=[touch1,touch2,touch3,touch4];
           touchvecmin=min(touchvec);
           touchvecmin_index=find(touchvec == touchvecmin);

           %Determine which pixel has the least amount of black pixels
           %next to it, then erase it

           erasepixel=touchvecmin_index(1);

           if erasepixel == 1
              x(k,m)=1;
           elseif erasepixel == 2
              x(k,m+1)=1;
           elseif erasepixel == 3
              x(k+1,m+1)=1;
           elseif erasepixel == 4
              x(k+1,m)=1;
           end

        end
     end
  end

  y=x;
```

%%%
%%%
%%%

- **detect_term_bif.m**

```
function [y_term,y_bif]=detect_term_bif(x)

%------------------------------------------------------------------------------------------------
% This function detects terminations and bifurcations in a thinned image
% ------------------------------------------------------------------------------------------------
% Input: Thinned image, in matrix form
%
% Output: y_term - matrix of data of same size as thinned image, with zeros
%                     at locations of terminations
%             y_bif - matrix of data of same size as thinned image, with zeros
%                     at locations of bifurcations
%
% Author: Graig Diefenderfer
% Date: May 2006
%------------------------------------------------------------------------------------------------

[row,col]=size(x);

y_term=ones(row,col);
y_bif=ones(row,col);
crossnum=zeros(row,col);

for k=2:row-1
   for m=2:col-1

      if x(k,m)==0
         crossnum(k,m)=(abs(x(k-1,m-1)-x(k-1,m))+abs(x(k-1,m)-x(k-1,m+1))...
            +abs(x(k-1,m+1)-x(k,m+1))+abs(x(k,m+1)-x(k+1,m+1))...
            +abs(x(k+1,m+1)-x(k+1,m))+abs(x(k+1,m)-x(k+1,m-1))...
            +abs(x(k+1,m-1)-x(k,m-1))+abs(x(k,m-1)-x(k-1,m-1)))/2;

         if crossnum(k,m)==1
            y_term(k,m)=0;

         elseif crossnum(k,m)>=3;
            y_bif(k,m)=0;
         end

      end
```

```
    end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **linetrace.m**

```
function y=linetrace(x,lineend,bif,tracedist)

%-----------------------------------------------------------------------------------------------
% This function traces lines to remove short spurs and short island segments
% -----------------------------------------------------------------------------------------------
% Input: x - thinned image
%          lineend - matrix of data of same size as thinned image, containing
%                      zeros at locations where traces will commence (usually
%                      terminations)
%          bif - matrix of data of same size as thinned image, containing zeros
%                at locations where traces will end (bifurcations when removing
%                spurs, terminations when removing short islands)
%          tracedist - maximum allowable trace length
%
% Output: Thinned image with unwanted segments removed
%
% Author: Graig Diefenderfer
% Date: May 2006
%-----------------------------------------------------------------------------------------------

[row,col]=size(x);
r=2;
c=2;
rnew=0;
cnew=0;
rold=0;
cold=0;
rtotal=[];
ctotal=[];
counter=0;
checkbif=1;

linelength=1;

for k=1:row
    for m=1:col
```

```
if lineend(k,m)==0

    linelength=1;
    r=k;
    c=m;
    rtotal=[];
    ctotal=[];
    checkbif=1;
    singlepoint=0;

    while (linelength < tracedist) && (checkbif==1) && (r>1) && …
            (r<=row-1) && (c>1) && (c<=col-1)

        x(r,c)=1;

        if x(r-1,c)==0
            rnew=r-1;
            cnew=c;
        elseif x(r,c+1)==0
            rnew=r;
            cnew=c+1;
        elseif x(r+1,c)==0
            rnew=r+1;
            cnew=c;
        elseif x(r,c-1)==0
            rnew=r;
            cnew=c-1;
        elseif x(r-1,c-1)==0
            rnew=r-1;
            cnew=c-1;
        elseif x(r-1,c+1)==0
            rnew=r-1;
            cnew=c+1;
        elseif x(r+1,c+1)==0
            rnew=r+1;
            cnew=c+1;
        elseif x(r+1,c-1)==0
            rnew=r+1;
            cnew=c-1;
        else            %it is a single isolated pixel
            rnew=r;
            cnew=c;
            singlepoint=1;
        end

        r=rnew;
```

```
            c=cnew;
            linelength=linelength+1;
            checkbif=bif(r,c);

            rtotal=[rtotal;rnew];
            ctotal=[ctotal;cnew];
        end

        if checkbif == 1    %Bifurcation not reached - replace points
            for h=1:length(rtotal)
                x(rtotal(h),ctotal(h))=0;
                x(k,m)=0;
            end
        end

        if singlepoint == 1     %delete isolated pixel
            x(k,m)=1;
        end
      end
   end
end

y=x;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **removehorizduplicate.m**

```
function y=removehorizduplicate(x)

%----------------------------------------------------------------------------------------------
% This function locates duplicate horizontal lines in thinned image and turns
% the lower of the two lines from black to white
% ----------------------------------------------------------------------------------------------
% Input: Thinned image
%
% Output: Thinned image with duplicate horizontal lines removed
%
% Author: Graig Diefenderfer
% Date: May 2006
%----------------------------------------------------------------------------------------------

[row,col]=size(x);

for k=5:row-5
```

```
for m=1:col-5

   scanright=0;
   scanleft=0;
   rightshift=1;
   leftshift=1;

   horizsum=sum(x(k,m:m+5));

   if horizsum == 0      %straight line, check for lines beneath

      horizsum1=sum(x(k+1,m:m+5));
      horizsum2=sum(x(k+2,m:m+5));
      horizsum3=sum(x(k+3,m:m+5));
      horizsum4=sum(x(k+4,m:m+5));
      horizsum5=sum(x(k+5,m:m+5));

      sumvector=[horizsum1,horizsum2,horizsum3,horizsum4,horizsum5];

      lineindex=find(sumvector == 0);

      if length(lineindex) > 0     %i.e. not an empty vector

         x(k+lineindex(1),m)=1;   %turn to white

         while scanright == 0
            scanright=x(k+lineindex(1),m+rightshift);

            if scanright == 0
               x(k+lineindex(1),m+rightshift)=1;
            end

            rightshift=rightshift+1;
         end


         while scanleft == 0
            scanleft=x(k+lineindex(1),m-leftshift);

            if scanleft == 0
               x(k+lineindex(1),m-leftshift)=1;
            end

            leftshift=leftshift+1;
         end
```

```
            end
        end
    end
end

y=x;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **removevertduplicate.m**

```
function y=removevertduplicate(x)

%-----------------------------------------------------------------------------------------------
% This function locates duplicate vertical lines in thinned image and
% deletes the unwanted segment
% -----------------------------------------------------------------------------------------------
% Input: Thinned image
%
% Output: Thinned image with duplicate vertical lines removed
%
% Functions Used: segmentlength.m, deletevertseg.m
%
% Author: Graig Diefenderfer
% Date: May 2006
%-----------------------------------------------------------------------------------------------

[row,col]=size(x);

[xlineend,xbif]=detect_term_bif(x);

counter=0;

for k=10:row-10
    for m=10:col-10

        scanup=0;
        scandown=0;
        upshift=1;
        downshift=1;

        vertsum=sum(x(k:k+10,m));

        if vertsum == 0     %vertical line, check for lines beside
```

```
        Rvertsum1=sum(x(k:k+10,m+1));
        Rvertsum2=sum(x(k:k+10,m+2));
        Rvertsum3=sum(x(k:k+10,m+3));
        Rsumvector=[Rvertsum1,Rvertsum2,Rvertsum3];
        Rlineindex=find(Rsumvector == 0);

        if length(Rlineindex) > 0       %i.e. not an empty vector

           counter=counter+1;

           length1=segmentlength(x,k,m);
           length2=segmentlength(x,k,m+Rlineindex(1));

           if (length1 < length2)         %delete the smaller segment
              x=deletevertseg(x,k,m);
           elseif (length2 < length1)
              x=deletevertseg(x,k,m+Rlineindex(1));
           end

           % when length1 == length2, segmentlength has reached the
           % maximum allowable count, and neither is a spurious line

        end
     end

  end
end

counter;
y=x;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **deletevertseg.m**

```
function y2=deletevertseg(x,r1,c1)

%-------------------------------------------------------------------------------------------------
% This function deletes unwanted duplicate vertical segments by tracing
% it downward and deleting pixel by pixel until end of segment is reached
% ------------------------------------------------------------------------------------------------
% Input: x - thinned image
%          r1 - row index of ridge at location where deletion begins
%          c1 - column index of ridge at location where deletion begins
%
```

```
% Output: Thinned image without duplicate vertical segment
%
% Author: Graig Diefenderfer
% Date: May 2006
%-------------------------------------------------------------------------------------------------

%Used in conjunction with removevertduplicate

keeptracing=1;
rnew=0;
cnew=0;

x(r1,c1)=1;   %Turn white

%move down to begin

if x(r1+1,c1)==0
   rnew=r1+1;
   cnew=c1;
   x(rnew,cnew)=1;
elseif x(r1+1,c1-1)==0
   rnew=r1+1;
   cnew=c1-1;
   x(rnew,cnew)=1;
elseif x(r1+1,c1+1)==0
   rnew=r1+1;
   cnew=c1+1;
   x(rnew,cnew)=1;
else
   keeptracing=0;
end

r=rnew;
c=cnew;

while keeptracing == 1

   x(r,c)=1;

   if x(r-1,c)==0
      rnew=r-1;
      cnew=c;
   elseif x(r,c+1)==0
      rnew=r;
      cnew=c+1;
   elseif x(r+1,c)==0
```

```
    rnew=r+1;
    cnew=c;
  elseif x(r,c-1)==0
    rnew=r;
    cnew=c-1;
  elseif x(r-1,c-1)==0
    rnew=r-1;
    cnew=c-1;
  elseif x(r-1,c+1)==0
    rnew=r-1;
    cnew=c+1;
  elseif x(r+1,c+1)==0
    rnew=r+1;
    cnew=c+1;
  elseif x(r+1,c-1)==0
    rnew=r+1;
    cnew=c-1;
  else
    keeptracing=0;
  end

  r=rnew;
  c=cnew;

end

y2=x;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **segmentlength.m**

```
function total=segmentlength(x,startingrow,startingcol)

%-------------------------------------------------------------------------------------------
% This function traces a segment until it reaches an end or arrives
% at the maximum trace distance, while maintaining a count of total
% segment length
% ------------------------------------------------------------------------------------------
% Input: x - thinned image
%        startingrow - row index to start trace
%        startingcol - column index to start trace
%
% Output: Number indicating total trace length
%
```

```
% Author: Graig Diefenderfer
% Date: May 2006
%-------------------------------------------------------------------------------------------------

%Used in conjunction with removevertduplicate, and begins by searching
%downward

total=1;
keeptracing=1;
rnew=0;
cnew=0;

x(startingrow,startingcol)=1;   %Turn white

%search down to begin

if x(startingrow+1,startingcol)==0
   rnew=startingrow+1;
   cnew=startingcol;
   x(rnew,cnew)=1;
elseif x(startingrow+1,startingcol-1)==0
   rnew=startingrow+1;
   cnew=startingcol-1;
   x(rnew,cnew)=1;
elseif x(startingrow+1,startingcol+1)==0
   rnew=startingrow+1;
   cnew=startingcol+1;
   x(rnew,cnew)=1;
else
   keeptracing=0;
end

r=rnew;
c=cnew;

while (keeptracing == 1) && (total < 40)

   x(r,c)=1;

   if x(r-1,c)==0
      rnew=r-1;
      cnew=c;
   elseif x(r,c+1)==0
      rnew=r;
      cnew=c+1;
   elseif x(r+1,c)==0
```

```
        rnew=r+1;
        cnew=c;
    elseif x(r,c-1)==0
        rnew=r;
        cnew=c-1;
    elseif x(r-1,c-1)==0
        rnew=r-1;
        cnew=c-1;
    elseif x(r-1,c+1)==0
        rnew=r-1;
        cnew=c+1;
    elseif x(r+1,c+1)==0
        rnew=r+1;
        cnew=c+1;
    elseif x(r+1,c-1)==0
        rnew=r+1;
        cnew=c-1;
    else
        keeptracing=0;
    end

    r=rnew;
    c=cnew;
    total=total+1;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **thinbwimage.m**

```
function y=thinbwimage(xin)

%------------------------------------------------------------------------------------------------
% This function is the main code for thinning via the central line
% method
% ------------------------------------------------------------------------------------------------
% Input: Black and White image in matrix form
%
% Output: Thinned image
%
% Author: Graig Diefenderfer (Developed from process described in
%                                    [Ahmed & Ward, 2002])
% Date: May 2006
%------------------------------------------------------------------------------------------------
```

```
%%%% Black pixel = 0
%%%% White pixel = 1

x=xin;
y=xin;
[row,col]=size(x);
stopscanmatrix=zeros(row,col);  %used in debugging
difference=1;

while(difference ~= 0)

   for k=3:row-3
      for m=3:col-3

         twover=0;
         twohor=0;
         stopscan=0;

         if x(k,m)==0

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check to see if the pixel belongs to two pixels width in vertical then
% horizontal direction

            if ( x(k-1,m)==1 && x(k+1,m)==0 && x(k+2,m)==1 ) || …
                  ( x(k-2,m)==1 && x(k-1,m)==0 && x(k+1,m)==1 )

               twover = 1;
            end

            if ( x(k,m-1)==1 && x(k,m+1)==0 && x(k,m+2)==1 ) || …
                  ( x(k,m-2)==1 && x(k,m-1)==0 && x(k,m+1)==1 )

               twohor = 1;
            end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Begin analysis for two pixel width in vertical direction

            if twover==1

               if ( x(k,m-1)==0 && x(k+1,m-1)==0 && x(k+1,m)==0 && …
                    x(k+1,m+1)==0 && x(k,m+1)==0 && x(k-1,m)==1 && x(k+2,m)==1 )

                  stopscan = 1;
```

```
        elseif (x(k,m-1)==0 && x(k-1,m-1)==0 && x(k-1,m)==0 && …
           x(k-1,m+1)==0 && x(k,m+1)==0 && x(k-2,m)==1 && x(k+1,m)==1 )

           y(k,m) = 1;     %delete pixel
           stopscan = 1;

        end

        if stopscan==0

           if( x(k+1,m)==0 && x(k+1,m-1)==0 && x(k+2,m-1)==0 && …
               x(k,m-1)==1 && x(k-1,m-1)==1 && x(k-1,m)==1 && …
               x(k-1,m+1)==1 && x(k,m+1)==1 && x(k+1,m+1)==1 && ...
               x(k+2,m+1)==1 && x(k+2,m)==1 )
              stopscan = 1;

           elseif (x(k+1,m)==0 && x(k+1,m+1)==0 && x(k+2,m+1)==0 && …
                    x(k+2,m)==1 && x(k+2,m-1)==1 && x(k+1,m-1)==1 && …
                    x(k,m-1)==1 && x(k-1,m-1)==1 && x(k-1,m)==1 && …
                    x(k-1,m+1)==1 && x(k,m+1)==1 )
              stopscan = 1;

           elseif ( x(k-1,m)==0 && x(k-1,m-1)==0 && x(k-2,m-1)==0 && …
                     x(k-2,m)==1 && x(k-2,m+1)==1 && x(k-1,m+1)==1 && …
                     x(k,m+1)==1 && x(k+1,m+1)==1 && x(k+1,m)==1 && …
                     x(k+1,m-1)==1 && x(k,m-1)==1 )
              stopscan = 1;

           elseif ( x(k-1,m)==0 && x(k-1,m+1)==0 && x(k-2,m+1)==0 && …
                     x(k-2,m)==1 && x(k-2,m-1)==1 && x(k-1,m-1)==1 && …
                     x(k,m-1)==1 && x(k+1,m-1)==1 && x(k+1,m)==1 && …
                     x(k+1,m+1)==1 && x(k,m+1)==1 )
              stopscan = 1;

           end

        end

     end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Begin analysis for two pixel width in horizontal direction

     if twohor==1
```

```
   if( x(k-1,m)==0 && x(k-1,m+1)==0 && x(k,m+1)==0 && …
      x(k+1,m+1)==0 && x(k+1,m)==0 && x(k,m-1)==1 && x(k,m+2)==1 )

      stopscan = 1;

   elseif ( x(k-1,m)==0 && x(k-1,m-1)==0 && x(k,m-1)==0 && …
       x(k+1,m-1)==0 && x(k+1,m)==0 && x(k,m-2)==1 && x(k,m+1)==1 )

      y(k,m) = 1;     %delete pixel
      stopscan = 1;

   end

   if stopscan==0

      if( x(k,m+1)==0 && x(k+1,m+1)==0 && x(k+1,m+2)==0 && …
          x(k,m+2)==1 && x(k-1,m+2)==1 && x(k-1,m+1)==1 && …
          x(k-1,m)==1 && x(k-1,m-1)==1 && x(k,m-1)==1 && …
          x(k+1,m-1)==1 && x(k+1,m)==1 )
         stopscan = 1;

      elseif ( x(k,m+1)==0 && x(k-1,m+1)==0 && x(k-1,m+2)==0 && …
                x(k,m+2)==1 && x(k+1,m+2)==1 && x(k+1,m+1)==1 && …
                x(k+1,m)==1 && x(k+1,m-1)==1 && x(k,m-1)==1 && …
                x(k-1,m-1)==1 && x(k-1,m)==1 )
         stopscan = 1;

      elseif ( x(k,m-1)==0 && x(k+1,m-1)==0 && x(k+1,m-2)==0 && …
               x(k,m-2)==1 && x(k-1,m-2)==1 && x(k-1,m-1)==1 && …
               x(k-1,m)==1 && x(k-1,m+1)==1 && x(k,m+1)==1 && …
               x(k+1,m+1)==1 && x(k+1,m)==1 )
         stopscan = 1;

      elseif ( x(k,m-1)==0 && x(k-1,m-1)==0 && x(k-1,m-2)==0 && …
                x(k,m-2)==1 && x(k+1,m-2)==1 && x(k+1,m-1)==1 && …
                x(k+1,m)==1 && x(k+1,m+1)==1 && x(k,m+1)==1 && …
                x(k-1,m+1)==1 && x(k-1,m)==1 )
         stopscan = 1;

      end

   end

end

stopscanmatrix(k,m)=stopscan;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Apply 21 rules to pixel

        if stopscan==0

          %obtain the value for each surrounding pixel
          %
          %    b1  b2  b3
          %    b8  x   b4
          %    b7  b6  b5
          %

          b1=x(k-1,m-1);
          b2=x(k-1,m);
          b3=x(k-1,m+1);
          b4=x(k,m+1);
          b5=x(k+1,m+1);
          b6=x(k+1,m);
          b7=x(k+1,m-1);
          b8=x(k,m-1);

%1
          if( b1==0 && b6==0 && b7==0 && b8==0 && b3==1 && b4==1 )
            y(k,m)=1;

%2
          elseif( b1==0 && b2==0 && b7==0 && b8==0 && b4==1 && b5==1 )
            y(k,m)=1;

%3
          elseif( b1==0 && b2==0 && b3==0 && b4==0 && b6==1 && b7==1 )
            y(k,m)=1;

%4
          elseif( b1==0 && b2==0 && b3==0 && b8==0 && b5==1 && b6==1 )
            y(k,m)=1;

%5
          elseif( b1==0 && b8==0 && b3==1 && b4==1 && b5==1 && b6==1 )
            y(k,m)=1;

%6
          elseif( b1==0 && b2==0 && b4==1 && b5==1 && b6==1 && b7==1 )
            y(k,m)=1;
```

```
%7
        elseif( b1==0 && b2==0 && b3==0 && b5==0 && b6==0 && …
              b7==0 && b8==0 && b4==1 )
           y(k,m)=1;

%8
        elseif( b1==0 && b2==0 && b3==0 && b4==0 && b5==0 && …
              b7==0 && b8==0 && b6==1 )
           y(k,m)=1;

%9
        elseif( b7==0 && b8==0 && b2==1 && b3==1 && b4==1 && b5==1 )
           y(k,m)=1;

%10
        elseif( b6==0 && b7==0 && b1==1 && b2==1 && b3==1 && b4==1 )
           y(k,m)=1;

%11
        elseif( b2==0 && b3==0 && b5==1 && b6==1 && b7==1 && b8==1 )
           y(k,m)=1;

%12
        elseif( b3==0 && b4==0 && b1==1 && b6==1 && b7==1 && b8==1 )
           y(k,m)=1;

%13
        elseif( b5==0 && b6==0 && b1==1 && b2==1 && b3==1 && b8==1 )
           y(k,m)=1;

%14
        elseif( b4==0 && b5==0 && b1==1 && b2==1 && b7==1 && b8==1 )
           y(k,m)=1;

%15
        elseif( b1==0 && b2==0 && b3==0 && b4==0 && b5==0 && …
              b6==0 && b7==0 && b8==1 )
           y(k,m)=1;

%16
        elseif( b1==0 && b3==0 && b4==0 && b5==0 && b6==0 && …
              b7==0 && b8==0 && b2==1 )
           y(k,m)=1;

%17
```

```
            elseif( b3==0 && b4==0 && b5==0 && b6==0 && b1==1 && b8==1 )
               y(k,m)=1;

%18
            elseif( b2==0 && b3==0 && b4==0 && b5==0 && b7==1 && b8==1 )
               y(k,m)=1;

%19
            elseif( b4==0 && b5==0 && b6==0 && b7==0 && b1==1 && b2==1 )
               y(k,m)=1;

%20
            elseif( b5==0 && b6==0 && b7==0 && b8==0 && b2==1 && b3==1 )
               y(k,m)=1;

%21
            elseif( b1==1 && b2==1 && b3==1 && b4==1 && b5==1 && …
                  b6==1 && b7==1 && b8==1 )
               y(k,m)=1;

            end

         end

      end        %closes initial 'if x(k,m)==0' loop

   end
end

difference=sum(sum(y-x));   %compare the difference between images
                  %If difference == 0, then no change has
                  %taken place and thinning is complete.
                  %The code will then exit the while loop
x=y;

end        %closes while loop

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Apply diagonal rules to thinned image

for k=3:row-3
   for m=3:col-3

      if y(k,m)==0

         %obtain the value for each surrounding pixel
```

```
        %
        %    p1  p2  p3
        %    p8  x   p4
        %    p7  p6  p5
        %

        p1=y(k-1,m-1);
        p2=y(k-1,m);
        p3=y(k-1,m+1);
        p4=y(k,m+1);
        p5=y(k+1,m+1);
        p6=y(k+1,m);
        p7=y(k+1,m-1);
        p8=y(k,m-1);


        if ( p2==0 && p8==0 && p4==1 && p5==1 && p6==1 )    %D1
           y(k,m)=1;
        elseif( p4==0 && p6==0 && p1==1 && p2==1 && p8==1 ) %D2
           y(k,m)=1;
        elseif( p2==0 && p4==0 && p6==1 && p7==1 && p8==1 ) %D3
           y(k,m)=1;
        elseif( p6==0 && p8==0 && p2==1 && p3==1 && p4==1 ) %D4
           y(k,m)=1;
        end

      end

   end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **get_minutiae_data.m**

```
function min_data2=get_minutiae_data(fingfile,thinningmethod)

%-------------------------------------------------------------------------------------------------
% This function will take an input fingerprint file and locate the
% terminations and bifurcations.  It will also plot the locations of
% these minutiae superimposed on the original black and white image.
% ------------------------------------------------------------------------------------------------
% Input: fingfile - filename of input fingerprint image, in string format
%           thinningmethod - thinning method to be used
%                                   (1=central line, 2=block filter)
```

```
%
% Output: Minutiae data in matrix format
%               column 1: row indices
%               column 2: column indices
%               column 3: angle orientation
%               column 4: type of minutiae  (1=termination, 2=bifurcation)
%
% Author: Graig Diefenderfer
% Date: May 2006
%----------------------------------------------------------------------------------------------------

orig=imread(fingfile);
bw=im2bw(orig(:,:,1),185/255);
bw1=bw-0;              %make it a double
bw1(1:18,:)=1;         %erase title
[rows,cols]=size(orig(:,:,1));

%figure(6),imshow(orig)

%Thin the fingerprint image
if thinningmethod==1
   z1=thinbwimage(bw1);
elseif thinningmethod==2
   z1=thin_blockfilter(bw1);
end
%figure(17),imshow(z1)

[term,bifur]=detect_term_bif(z1);
z1=linetrace(z1,term,term,17);  %remove short island segments
z1=linetrace(z1,term,bifur,8); %remove spurs
%figure(18),imshow(z1)

[term,bifur]=detect_term_bif(z1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% isolate good portion of fingerprint via an ellipse

ellips=ones(rows,cols);
[blrow,blcol]=find(z1==0);  %locate all black pixels
rowrange=max(blrow)-min(blrow);
colrange=max(blcol)-min(blcol);
rowcenter=min(blrow)+ (rowrange/2);
colcenter=min(blcol)+ (colrange/2);
ellipse_a=0.47*rowrange;
ellipse_b=0.43*colrange;
```

```
for k=1:rows
   for m=1:cols

      ellipsecalc=((k-rowcenter)^2)/(ellipse_a^2)+((m-colcenter)^2)/(ellipse_b^2);

      if ellipsecalc < 1      %within ellipse, turn to black
         ellips(k,m)=0;
      end
   end
end

%figure(19),imshow((ellips+z1)/2)

x_term1=((ellips+term)/2);
x_term=im2bw(x_term1,0.1);
x_term=x_term-0;    %make it a double

x_bifur1=((ellips+bifur)/2);
x_bifur=im2bw(x_bifur1,0.1);
x_bifur=x_bifur-0;  %make it a double

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% get indices for terminations and bifurcations

[term_row,term_col]=find(x_term==0);
termpoints=length(term_row);
[bifur_row,bifur_col]=find(x_bifur==0);
bifurpoints=length(bifur_row);

term_angle = determine_term_angles(z1,term_row,term_col);

term_data=zeros(termpoints,3);
term_data(:,1)=term_row;
term_data(:,2)=term_col;
term_data(:,3)=term_angle;

%The terminations of the inverted image correspond to the bifurcations in
%the original image

bwinvert=abs(1-bw);         %Invert original image
%Thin the inverted fingerprint image
if thinningmethod==1
   thininvert=thinbwimage(bwinvert);
elseif thinningmethod==2
   thininvert=thin_blockfilter(bwinvert);
end
```

```
[term_inv,bifur_inv]=detect_term_bif(thininvert);
term_inv=((term_inv+ellips)/2);  %isolate good region of fingerprint
term_inv=im2bw(term_inv,0.1);
term_inv=term_inv-0;     %make it a double
[term_inv_row,term_inv_col]=find(term_inv==0);

bifur_data=determine_bif_angles(thininvert, bifur_row, bifur_col, term_inv_row,
term_inv_col);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% combine terminations and bifurcations into one matrix of data
% column 1: row indices
% column 2: column indices
% column 3: angle orientation
% column 4: type of minutiae  (1=termination, 2=bifurcation)

min_data=zeros(termpoints+bifurpoints,4);
min_data(1:termpoints,4)=1; %a 1 in the fourth column indicates termination
min_data(1:termpoints,1:3)=term_data;
min_data(termpoints+1:termpoints+bifurpoints,4)=2; %a 2 indicates bifurcation
min_data(termpoints+1:termpoints+bifurpoints,1:3)=bifur_data;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% remove terminations at edges of image

min_data2=remove_edge_term(min_data,z1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% plot minutiae, with line indicating angle orientation

min_display=plot_minutiae(min_data2,rows,cols);
figure(20),imshow(min_display)
figure(21),imshow((min_display/1.5)+(bw1/5))


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **determine_term_angles.m**

```
function y_angle = determine_term_angles(x,rowindex,colindex)

%-------------------------------------------------------------------------------------------
% This function computes the angles of terminations in a thinned image
% -------------------------------------------------------------------------------------------
```

```
% Input: x - thinned fingerprint image
%         rowindex - row indices of terminations
%         colindex - column indices of terminations
%
% Output: Corresponding orientation angles of each termination
%
% Author: Graig Diefenderfer
% Date: May 2006
%-------------------------------------------------------------------------------------------------

[row,col]=size(x);
numpoints=length(rowindex);
y_angle=zeros(numpoints,1);

maxtrace=5;
totaltrace=0;
r=0;
c=0;
rnew=0;
cnew=0;

for k=1:numpoints

   r=rowindex(k);
   c=colindex(k);
   totaltrace=0;

   while(totaltrace < maxtrace)

      x(r,c)=1;

      if x(r-1,c)==0
         rnew=r-1;
         cnew=c;
      elseif x(r,c+1)==0
         rnew=r;
         cnew=c+1;
      elseif x(r+1,c)==0
         rnew=r+1;
         cnew=c;
      elseif x(r,c-1)==0
         rnew=r;
         cnew=c-1;
      elseif x(r-1,c-1)==0
         rnew=r-1;
         cnew=c-1;
```

```
    elseif x(r-1,c+1)==0
        rnew=r-1;
        cnew=c+1;
    elseif x(r+1,c+1)==0
        rnew=r+1;
        cnew=c+1;
    elseif x(r+1,c-1)==0
        rnew=r+1;
        cnew=c-1;
    end

    r=rnew;
    c=cnew;
    totaltrace=totaltrace+1;

end   %at the end of the while loop, r and c represent the indices
      %that will be used to calculate the angle of the termination

r1=rowindex(k);
c1=colindex(k);
r2=r;
c2=c;

if(c1==c2 && r1>r2)
    y_angle(k)=270;

elseif(c1==c2 && r2>r1)
    y_angle(k)=90;

elseif(r1==r2 && c1>c2)
    y_angle(k)=0;

elseif(r1==r2 && c2>c1)
    y_angle(k)=180;

elseif(r1>r2 && c1>c2)
    y_angle(k)=360-(atan((r1-r2)/(c1-c2))*180/pi);

elseif(c1>c2 && r2>r1)
    y_angle(k)=90-(atan((c1-c2)/(r2-r1))*180/pi);

elseif(r1>r2 && c2>c1)
    y_angle(k)=180+(atan((r1-r2)/(c2-c1))*180/pi);

elseif(c2>c1 && r2>r1)
    y_angle(k)=90+(atan((c2-c1)/(r2-r1))*180/pi);
```

```
   end

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **determine_bif_angles.m**

```
function y=determine_bif_angles(xinv,biforow,bifocol,termirow,termicol)

%-------------------------------------------------------------------------------------------------
% This function computes the angles of bifurcations in a thinned image
% ------------------------------------------------------------------------------------------------
% Input: xinv - thinned inverted fingerprint image
%        biforow - bifurcations from original image, row indices
%        bifocol - bifurcations from original image, column indices
%        termirow - terminations from inverted image, row indices
%        termicol - terminations from inverted image, column indices
%
% Output: Matrix of data
%             column1 - bifurcation row indices
%             column2 - bifurcation column indices
%             column3 - bifur angles assigned to row/col coordinates
%
% Author: Graig Diefenderfer
% Date: May 2006
%-------------------------------------------------------------------------------------------------

invtermangles=determine_term_angles(xinv,termirow,termicol);
invtermpoints=length(termirow);
invtermdata=zeros(invtermpoints,3);
invtermdata(:,1)=termirow;
invtermdata(:,2)=termicol;
invtermdata(:,3)=invtermangles;

bifpoints=length(biforow);
bifdata=zeros(bifpoints,3);
bifdata(:,1)=biforow;
bifdata(:,2)=bifocol;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Compare position locations between terminations of inverted image to
%bifurcations of original image.  If a termination of the inverted image
```

```
%lies close to a bifurcation in the original image, assign the angle of
%the termination as the angle of the bifurcation.  If a bifurcation does
%not match any terminations, keep its angle at zero.

for k=1:bifpoints

    lowestdistance=100; %reset lowestdistance variable

    for m=1:invtermpoints

        rowdif=abs(bifdata(k,1)-invtermdata(m,1));
        coldif=abs(bifdata(k,2)-invtermdata(m,2));
        absolutedistance = sqrt(rowdif^2+coldif^2);

        if (absolutedistance <= 15) && (absolutedistance < lowestdistance)

            bifdata(k,3)=invtermdata(m,3);  %assign angle value

            lowestdistance=absolutedistance;    %change lowestdistance
        end

    end
end

y=bifdata;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **plot_minutiae.m**

```
function y=plot_minutiae(mdata,row,col)

%------------------------------------------------------------------------------------------------
% This function generates a matrix that highlights the minutiae along
% with their orientation angle.  Squares represent terminations and
% diamonds represent bifurcations.
% ------------------------------------------------------------------------------------------------
% Input: mdata - matrix of input minutiae data
%                   column 1: row indices
%                   column 2: column indices
%                   column 3: angle orientation
%                   column 4: type of minutiae  (1=termination, 2=bifurcation)
%             row - number of rows in original image
%             col - number of columns in original image
```

```
%
% Output: Matrix of data ready to be plotted
%
% Author: Graig Diefenderfer
% Date: May 2006
%-------------------------------------------------------------------------------------------------

[nrows,ncols]=size(mdata);

stdline=zeros(15,15);
stdline(8,8:15)=1;

% draw square at terminations and diamond at bifurcations

squ=ones(5,5);
squ(1,1:5)=0;
squ(2:4,5)=0;
squ(5,1:5)=0;
squ(2:4,1)=0;

diamond=ones(5,5);
diamond(1,3)=0;
diamond(2,2)=0;
diamond(3,1)=0;
diamond(4,2)=0;
diamond(5,3)=0;
diamond(4,4)=0;
diamond(3,5)=0;
diamond(2,4)=0;

y1=ones(row,col);
y2=zeros(row,col);

for k=1:nrows   %nrows is equal to the total number of minutiae
   rindex=mdata(k,1);
   cindex=mdata(k,2);
   newline=zeros(row,col);  %matrix of entire figure, will have ones only where new line
                            %will be added

   %draw squares and diamonds
   if mdata(k,4)==1    %termination
      y1(rindex-2:rindex+2,cindex-2:cindex+2)=squ;

   elseif mdata(k,4)==2    %bifurcation
      y1(rindex-2:rindex+2,cindex-2:cindex+2)=diamond;
```

```
    end

    drawline=stdline;
    drawline=imrotate(stdline,mdata(k,3));  %has ones where line is

    [rline,cline]=size(drawline);
    midrow=ceil(rline/2);
    midcol=ceil(cline/2);

    newline(rindex-midrow+1:rindex+(rline-midrow),cindex-midcol+1:cindex+ …
            (cline-midcol))=drawline;

    y2=y2+newline;

end

%y2 has positive integer values at locations where a line should be drawn
y2=im2bw(y2,0.1);
y2=abs(1-y2);    %invert image, to have lines be drawn black

y=(y1+y2)/2;
y=im2bw(y,0.9);
y=y-0;  %make it a double


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **minutiae_match.m**

```
function ym=minutiae_match(idata,tdata)

%-------------------------------------------------------------------------------------------
% This function takes two matrices of data, and computes the overall matching score
% between the two sets.
% ------------------------------------------------------------------------------------------
% Input: idata - Input fingerprint data
%          tdata - Template fingerprint data
%
%                  column 1: row indices
%                  column 2: column indices
%                  column 3: angle orientation
%                  column 4: type of minutiae  (1=termination, 2=bifurcation)
%
% Output: Matching score between the two matrices of minutiae data
%
```

```
% Functions Used: match_score.m
%
% Author: Graig Diefenderfer (Developed from process described in
%                             [Luo, Tian, & Wu, 2000])
% Date: May 2006
%-------------------------------------------------------------------------------------------------

[idatarow,idatacol]=size(idata);
[tdatarow,tdatacol]=size(tdata);
matchingscore=zeros(tdatarow,idatarow);

%number of minutiae points for input and template is equal to the number of
%rows in idata and tdata, respectively

%create a matrix containing all the possible rotation values
rotatevalues=zeros(tdatarow,idatarow);
for k=1:tdatarow
   for m=1:idatarow
      rotatevalues(k,m)=tdata(k,3)-idata(m,3);
   end
end

%convert each minutiae point to polar coordinates with respect to the
%reference minutiae in each case
for k=1:tdatarow
   for m=1:idatarow

      tdatapolar=zeros(tdatarow,tdatacol);
      idatapolar=zeros(idatarow,idatacol);

      tref=tdata(k,:);
      iref=idata(m,:);

      tdatapolar(:,1)=sqrt((tdata(:,1)-tref(1)).^2 + (tdata(:,2)-tref(2)).^2);
      tdatapolar(:,2)=atan2(tref(1)-tdata(:,1),tdata(:,2)-tref(2)) * 180/pi;
      %rows give y displacement, cols give x diplacement
      tdatapolar(:,2)=mod(tdatapolar(:,2),360);   %get angles between 0 and 360
      tdatapolar(:,3)=tdata(:,3)-tref(3);
      tdatapolar(:,3)=mod(tdatapolar(:,3),360);    %get angles between 0 and 360
      tdatapolar(:,4)=tdata(:,4);

      idatapolar(:,1)=sqrt((idata(:,1)-iref(1)).^2 + (idata(:,2)-iref(2)).^2);
      idatapolar(:,2)=(atan2(iref(1)-idata(:,1),idata(:,2)-iref(2)) * 180/pi) + …
                        rotatevalues(k,m);
      idatapolar(:,2)=mod(idatapolar(:,2),360);   %get angles between 0 and 360
      idatapolar(:,3)=idata(:,3)-iref(3);
```

```
        idatapolar(:,3)=mod(idatapolar(:,3),360);   %get angles between 0 and 360
        idatapolar(:,4)=idata(:,4);

        matchingscore(k,m)=compute_match_score(tdatapolar,idatapolar);

    end
end

maxmatchingscore=max(max(matchingscore));
maxminutiae=max(idatarow,tdatarow);     %get maximum minutiae values between input
and template

ym=maxmatchingscore/maxminutiae;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **compute_match_score.m**

```
function mscore=compute_match_score(tpol,ipol)

%-----------------------------------------------------------------------------------------------
% This function is called from the minutiae_matching function.  It
% compares the data and computes their matching score.
% -----------------------------------------------------------------------------------------------
% Input: tpol - template data points, in polar coordinates
%         ipol - input data points, in polar coordinates
%                  column 1: radial distance
%                  column 2: radial angle
%                  column 3: minutiae orientation
%                  column 4: type of minutiae  (1=termination, 2=bifurcation)
%
% Output: Matching score for one combination of reference points
%
% Author: Graig Diefenderfer (Developed from process described in
%                                  [Luo, Tian, & Wu, 2000])
% Date: May 2006
%-----------------------------------------------------------------------------------------------

tpoints=size(tpol,1);   %number of rows=number of template points
ipoints=size(ipol,1);   %number of rows=number of input points

%use fixed size bounding box for all template points
radsize=zeros(tpoints,1);
angsize=zeros(tpoints,1);
```

```
radsize(:)=6;
angsize(:)=10;

radlow=-radsize./2;
radhigh=radsize./2;

anglow=-angsize./2;
anghigh=angsize./2;

epsillon=10;
mscore=0;

for kk=1:tpoints
   for mm=1:ipoints

      rdiff=tpol(kk,1)-ipol(mm,1);
      ediff=tpol(kk,2)-ipol(mm,2);
      thetadiff=tpol(kk,3)-ipol(mm,3);

      if ((radlow(kk) < rdiff) && (rdiff < radhigh(kk)) && (anglow(kk) < ediff) && …
         (ediff < anghigh(kk)) && (abs(thetadiff) < epsillon) && …
         (tpol(kk,4)==ipol(mm,4)))

         mscore=mscore+1;
         tpol(kk,4)=3;   %Change type of minutiae, indicating it has been used in a match
                         %already
      end

   end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **remove_edge_term.m**

```
function ym=remove_edge_term(mdata,xthin)

%-----------------------------------------------------------------------------------------------------
% This function removes terminations that lie near the edges of the
% fingerprint image.  These terminations are not true minutiae, but they
% appear as terminations because the ridges are cutoff along the edges of
% the image.
% -----------------------------------------------------------------------------------------------------
```

```
% Input: mdata - matrix of input minutiae data
%                   column 1: row indices
%                   column 2: column indices
%                   column 3: angle orientation
%                   column 4: type of minutiae  (1=termination, 2=bifurcation)
%         xthin - thinned fingerprint image
%
% Output: Matrix of minutiae data with incorrect terminations along the
%          edges removed
%
% Author: Graig Diefenderfer
% Date: May 2006
%-------------------------------------------------------------------------------------------------

ym=mdata;
[mrows,mcols]=size(mdata);
searchlength=20;
remterm=0;

rowcount=1;     %represents minutiae data number
checktype=mdata(rowcount,4);

[thinrow,thincol]=size(xthin);

while checktype==1 && rowcount<=mrows  %only use terminations

  mangle=(mdata(rowcount,3));
  rstart=mdata(rowcount,1);
  cstart=mdata(rowcount,2);

  if ((mangle >= 45) && (mangle < 135)) || ((mangle >= 225) && (mangle <315))
     %search left and right, ensuring that the search window stays within
     %the dimensions of the thinned image, and the search length doesn't
     %exceed the value of searchlength defined previously
     sleft=min((cstart-1),searchlength);
     sright=min((thincol-cstart),searchlength);
     leftpoints=xthin(rstart,cstart-sleft:cstart-1);
     rightpoints=xthin(rstart,cstart+1:cstart+sright);

     flipleft=1-leftpoints;   %now, white pixels = 0, black pixels = 1
     flipright=1-rightpoints;

     %if all elements in flipright or flipleft are white (0), then this
     %termination is on the edge of the image and should be removed

     if (sum(flipleft)==0) || (sum(flipright)==0)
```

```
        ym(rowcount,4)=0;    %change termination type, indicating it should be deleted
      end

   else     %search up and down, ensuring that the search window stays
      %within the dimensions of the thinned image, and the search length
      %doesn't exceed the value of searchlength defined previously
      sup=min((rstart-1),searchlength);
      sdown=min((thinrow-rstart),searchlength);
      uppoints=xthin(rstart-sup:rstart-1,cstart);
      downpoints=xthin(rstart+1:rstart+sdown,cstart);

      flipup=1-uppoints;       %now, white pixels = 0, black pixels = 1
      flipdown=1-downpoints;

      if (sum(flipup)==0) || (sum(flipdown)==0)
         ym(rowcount,4)=0;    %change termination type, indicating it should be deleted
      end

   end

   rowcount=rowcount+1;
   if rowcount <= mrows
       checktype=mdata(rowcount,4);
   end
end

delrow=find(ym(:,4)==0);
ym(delrow,:)=[];     %delete rows with 0 in 4th column


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

- **generate_templates.m**

```
%-------------------------------------------------------------------------------------------
% This program generates the template data for the database of synthetic
% images
% ------------------------------------------------------------------------------------------
% Output: Saved template data in matrix format
%
% Author: Graig Diefenderfer
% Date: May 2006
%-------------------------------------------------------------------------------------------

clear
```

```
t1_total=0;
t2_total=0;

for k=0:59
   k
   if k < 10   %number is only one character
      newfilename=['s000',num2str(k),'_1.bmp'];
   else        %number is two characters
      newfilename=['s00',num2str(k),'_1.bmp'];
   end

   %Perform central thinning
   tic
   templatedata=get_minutiae_data(newfilename,1);
   save([newfilename(1:5),'_t.mat'],'templatedata')
   t1=toc;
   t1_total=t1_total+t1;   %Track total elapsed time for central thinning

   %Perform boundary block thinning
   tic
   templatedata=get_minutiae_data(newfilename,2);
   save([newfilename(1:5),'_t2.mat'],'templatedata')
   t2=toc;
   t2_total=t2_total+t2;   %Track total elapsed time for block thinning

end
```

LIST OF REFERENCES

Ahmed, M., & Ward, R. (2002). A rotation invariant rule-based thinning algorithm for character recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 24, 1672-1678*.

Boulgouris, N.V., Hatzinakos, D., & Plataniotis, K.N. (2005). Gait recognition: a challenging signal processing technology for biometric identification. *IEEE Signal Processing Magazine, 22, 78-90.*

Daugman, J. (2004). How iris recognition works. *IEEE Transactions on Circuits and Systems for Video Technology, 14, 21-30.*

Faundez-Zanuy, M. (2005). Biometric verification by means of hand geometry. *39th Annual 2005 International Carnahan Conference on Security Technology, 61-67.*

Faundez-Zanuy, M., & Monte-Moreno, E. (2005). State-of-the-art in speaker recognition. *IEEE Aerospace and Electronic Systems Magazine, 20, 7-12.*

Gao, Y., & Leung, M.K.H. (2002). Face recognition using line edge map. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 24, 764-779.*

International Biometric Group. (2003). *The Henry classification system*. Retrieved May 23, 2006, from http://www.biometricgroup.com/Henry%20Fingerprint%20Classification.pdf

Ito, K., Morita, A., Aoki, T., Higuchi, T., Nakajima, H., & Kobayashi, K. (2005). A fingerprint recognition algorithm using phase-based image matching for low-quality fingerprints. *Proceedings of IEEE International Conference on Image Processing, 2, 33-36.*

Jain, A., Hong. L., & Bolle, R. (1997). On-line fingerprint verification. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 19, 302-314.*

Jain, A.K., Pankanti, S., Prabhakar, S., Hong, L., & Ross, A. (2004). Biometrics: a grand challenge. *17th International Conference on Pattern Recognition, 2, 935-942.*

Jain, A.K., Prabhakar, S., Hong, L., & Pankanti, S. (2000). Filterbank-based fingerprint matching. *IEEE Transactions on Image Processing, 9, 846-859.*

Jain, A.K., Ross, A., & Prabhakar, S. (2004). An introduction to biometric recognition. *IEEE Transactions on Circuits and Systems for Video Technology, 14, 4-20.*

Ko, T. (2005). Multimodal biometric identification for large user population using fingerprint, face and iris recognition. *Proceedings of the 34th Applied Imagery and Pattern Recognition Workshop, 218-223.*

Luo, X., Tian, J., & Wu, Y. (2000). A minutia matching algorithm in fingerprint verification. *15th International Conference on Pattern Recognition, 4, 833-836.*

Mainguet, J.F. (2006). *Cellphones & PDAs with a built-in fingerprint sensor.* Retrieved April 29, 2006, from http://perso.wanadoo.fr/fingerchip/biometrics/types/fingerprint_products_pdaphones.htm

Maltoni, D., Maio, D., Jain, A.K., & Prabhakar, S. (2003). *Handbook of fingerprint recognition*. New York: Springer.

Patil, P.M., Suralkar, S.R., & Sheikh, F.B. (2005). Rotation invariant thinning algorithm to detect ridge bifurcations for fingerprint identification. *Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence, 634-641.*

Pentland, A., & Choudhury, T. (2000). Face recognition for smart environments. *Computer, 33, 50-55.*

Seow, B.C., Yeoh, S.K., Lai, S.L., & Abu, N.A. (2002). Image based fingerprint verification. *Student Conference on Research and Development, 58-61.*

Tico, M., Immonen, E., Rämö, P., Kuosmanen, P., & Saarinen, J. (2001). Fingerprint recognition using wavelet features. *IEEE International Symposium on Circuits and Systems, 2, 21-24.*

Woodward, J.D. (1997). Biometrics: privacy's foe or privacy's friend? *Proceedings of the IEEE, 85, 1480-1492.*

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Jeffrey B. Knorr
   Chairman, Department of Electrical
   and Computer Engineering
   Naval Postgraduate School
   Monterey, California

4. Professor Monique P. Fargues
   Naval Postgraduate School
   Monterey, California

5. Professor Roberto Cristi
   Naval Postgraduate School
   Monterey, California

6. ENS Graig T. Diefenderfer
   Bear, Delaware